Proyecto Doctoris

Sistemas Informáticos

2008 - 2009

Facultad de Informática,
Universidad Complutense de Madrid

Autores: Camilo Andrés Benito Rojas
Dulce María Valerón Almazán
Director Proyecto: Juan Pavón Mestras

Índice

Introducción 3

Motivación del Proyecto 3
Descripción 4
Agradecimientos 5
Propósito General y Definición de Objetivos 6
Bases Tecnológicas 6
Metodología 6

Desarrollo 7

Requisitos 7
Casos de uso 8
Riesgos 8
Diseño UML 9
Diseño de la interfaz 11
Diseño de la BD 15
Arquitectura 17
Arquitectura de la aplicación. 21
Arquitectura ICARO 27

Implementación 29

Pautas de Codificación 29
Organización del código 30
Interfaz 30
Estructura de paquetes. Implementación de agentes y recursos. 32
Base de datos 39
Seguridad y confidencialidad 40

Pruebas 42

Experimentación 43

Comunicación externa con ICARO 43
Separación de Agentes y Recursos 43
Versión portable 44

Resultados y conclusiones 46

Métricas. 46
Problemas encontrados y solución adoptada 47
Versiones futuras 50

Bibliografía 51

Glosario 51

Anexo I: Casos de uso 54

Anexo II: Métricas Investigación 65

Anexo III: Pruebas 66

Índice palabras clave

- Sistema multiagente.
- Infraestructura ICARO-T.
- Gestión de consultas medicas.
- Base de datos MySQL.
- Agentes y Recursos.

Introducción

Motivación del Proyecto

La idea de este proyecto surge de comprobar que a pesar de existir mucho software destinado a la medicina, la mayoría son bastante inflexibles o demasiado específicos. Además, no ofrecen servicios aplicables a las nuevas tecnologías de hoy en día como son el uso del web o el móvil.

A raíz de ello, se pensó en diseñar un sistema de gestión de consulta médica capaz de compatibilizar aplicación software, aplicación web, y aplicación móvil conjuntamente. Todo ello además con un diseño que facilite las adaptaciones a las distintas especialidades médicas.

Se trata de una aplicación muy completa y versátil que resultaría de gran utilidad para este sector profesional, permitiendo tener todas las interfaces centralizadas en la misma aplicación. Novedoso para lo que se estila actualmente en el mercado de este tipo de software.

Descripción

Este proyecto implementa un sistema de gestión de consulta médica multiplataforma, adaptable fácilmente a los distintos tipos de especialidades médicas y aplicaciones futuras. Se han seguido dos pautas fundamentales en el diseño.

La primera diseñar un sistema donde la funcionalidad sea independiente de la interfaz sobre la que se aplica. También conocido como Modelo - Vista - Controlador.

Esto permite que la aplicación sea adaptable a cualquier tipo de interfaz de manera sencilla sin suponer grandes cambios en la implementación. Además de la reutilización de código de manera eficiente.

En segundo lugar elaborar una implementación distribuida, dividida en distintos componentes para aumentar la flexibilidad en las posibles adaptaciones posteriores.

Al dividir la aplicación los cambios que se realicen solo afectan a una parte muy concreta del código. Fácil de localizar y depurar. Y lo nuevo que se añada será siempre independiente a lo ya existente.

Para llevar a cabo todo lo arriba mencionado se ha usado la herramienta ICARO-T.

ICARO-T es una Infraestructura Ligera de Componentes Software Java basada en Agentes y Recursos y Organizaciones para el desarrollo de aplicaciones distribuidas. Muy útil cuando se trata de construir un sistema distribuido como el que se pretendía en este caso.

This project implements a medical clinic management system that can be applied to any kind of interface (computer, mobile, web) and can adapt itself easily to changes. There are two basic guidelines followed in the design of the project.

The first one is to design a system where the functionality is independent of the interface to which it is applied. This is generally known as a Model View Controller pattern.

This allows the creation of an application which can be adapted to any system in a relatively simple way, without making big changes to the implementation of application. Another advantage is that the code can be reused efficiently.

The second guideline followed is to design a distributed implementation, separated in individual components, to make the application more flexible for possible future uses.

By doing this division, changes done in one component won't affect the correct operation of another one. Any change in the code will be done in a localized part of the code which makes it easier to debug.

A multiagent system was used in order to accomplish all that was mentioned above. More specifically, a multiagent infrastructure call ICARO-T is the base of the system.

ICARO-T is a Light Component Java Software Infrastructure based on Agents, Resources and Organizations for the development of distributed applications. It is a very useful tool when building a distributed system, which is what is pretended in this case.

Agradecimientos

En primer lugar queremos manifestar nuestro especial agradecimiento a Juan Pavón Mestras, director del proyecto, por brindarnos la oportunidad de realizar este proyecto y por su apoyo durante todo el proceso de desarrollo.

A Francisco J. Garijo, al que consideramos nuestro co-tutor, por su paciencia y disponibilidad permanente a resolvernos dudas de ICARO-T.

A los amigos, por el clima agradable que hemos creado entre todos durante estos años, y que siempre hacen de una carrera universitaria algo más agradable.

Y por ultimo, pero no por ello menos importante, a nuestras familias, que nos han apoyado, aconsejado, soportado y financiado a lo largo no solo del proyecto sino de toda la carrera.

Propósito General y Definición de Objetivos

El principal objetivo de este proyecto es el diseño de un sistema multiagente que sirva para poner de relieve las ventajas y desventajas que aporta la implementación de un sistema multiagente respecto a las clásicas orientadas a objetos.

Bases Tecnológicas

Uno de los objetivos del proyecto es el desarrollo de la aplicación usando un sistema multiagente. Para ello se ha utilizado una infraestructura llamada *ICARO-T* que proporciona una base sobre la que construir el sistema.

Por otro lado, al mismo tiempo que se desarrolla la aplicación médica usando ICARO-T, también se va probando el funcionamiento de esta infraestructura, ya que aún se encuentra en fase de desarrollo. Con esto se consigue probar el funcionamiento de ICARO-T en una aplicación real y así poder detectar fallos o posibles mejoras. En otras palabras, el trabajo hecho durante la realización del proyecto permitirá dar un feedback a los desarrolladores de ICARO-T una vez finalizado el desarrollo de la aplicación.

Aparte de ICARO-T se han usado otras tecnologías para facilitar el desarrollo de la aplicación.

- Lenguaje de programación: Java
- Interfaz gráfica: SWT
- Documentación: JavaDoc
- Base de datos: MySQL y JDBC

Metodología

Nada mas empezar con este proyecto, tuvimos un considerable periodo de investigación. Inicialmente basado en el concepto de multiagente y posteriormente de forma mas extensa, en la comprensión y uso de la plataforma ICARO-T.

Entre medias de todo esto, se empezaron a elaborar los requisitos de la aplicación, los casos de uso y a partir de estos los diagramas de secuencia de la aplicación.

Una vez que se tuvieron las ideas un poco claras, se diseñó la estructura general del sistema en función de los agentes y recursos que iba a necesitar nuestra aplicación. A continuación, dividimos el trabajo asignando a cada miembro del grupo la implementación de un conjunto de agentes.

Conociendo la ventaja de gran flexibilidad y modularidad que aporta este tipo de arquitectura, creamos un repositorio de código on-line sobre el que compactar el trabajo, sin que en ningún momento se dieran problemas de conflicto.

A medida que se completaba nueva funcionalidad, se añadían nuevas pruebas al banco de pruebas, de forma que los resultados de estas quedaban grabados para la posterior comprobación o corrección.

Desarrollo

Para poder definir con exactitud la funcionalidad del proyecto y su alcance, se definieron los requisitos funcionales y no funcionales que debía cumplir la aplicación. A raíz de aquí todo lo que se diseño a continuación se basó en estas directrices iniciales.

Requisitos

Para elaborar un diseño eficiente se contacto con posibles clientes relacionados con la medicina interesados en el proyecto que nos ayudaron a definir los requisitos funcionales y no funcionales indispensables para este tipo de sistemas.

- Funcionales:
    - Agenda de pacientes (multi-médico)
        - Petición hora
        - Estado consulta actual (En consulta, en sala de espera, tiempo estimado de entrada/ retraso)
        - Creación /consulta/modificación fichas.
        - Configuración de la agenda.
        - Búsqueda de visitas por paciente
    - Gestión de la visita
        - Creación / modificación / consulta del historial
        - Módulos rellenables específicos de cada médico e intercambiables (analítica, fotos, diagnóstico, recetas)
        - Motivo de consulta (corto)
        - Descripción de la enfermedad actual
        - Antecedentes
        - Exploración física
        - Analítica (laboratorio, ecografía, rayos, tele-consulta), documentos, imágenes
        - Diagnóstico
        - Tratamientos
    - Recetas (creación, modificación, impresión)
    - Consentimientos (creación, modificación, impresión)
    - Búsquedas por medicamento / enfermedad
    - Gestiones administrativas
        - Creación profesionales

- No funcionales:
    - Sistema operativo: Compatibilidad con Windows.
    - Memoria Ram: Mínimo 256MB.
    - Espacio en disco: Inferior a 50MB.
    - Ancho de banda: Banda ancha de 1MB mínimo.
    - Navegador Internet Explorer, Firefox o compatible.
    - Software adicional: Siempre que no suponga un coste adicional.
    - Tiempo de respuesta (tiempo que pasa desde que se envía una orden y se recibe la respuesta): No superior a 2 segundos.
    - Confiabilidad: En ningún caso peligro en la integridad de los datos.
    - Escalabilidad (capacidad del sistema informático de cambiar su tamaño o configuración para adaptarse a las circunstancias cambiantes): Adaptación eficiente de la aplicación según el crecimiento de la empresa.
    - Flexibilidad: Capacidad para adaptarse a las distintas especialidades medicas de forma sencilla.

Casos de uso

Una vez completados los requisitos, se definieron los casos de uso para tener claro los distintos escenarios de los que constaba la aplicación. Gracias a ellos empezamos a asignar los agentes y recursos que necesitábamos y a intuirse el diseño inicial de la interfaz de usuario.

Para mas detalle se puede consultar el Anexo I Pág. 51.

Riesgos

Los riesgos se estuvieron definiendo y actualizando a lo largo de todo el proyecto. Aquí adjuntamos los más representativos.

Riesgo	Accion	Probabilidad	Consecuencias
Fallo de hardware	Buscar un ordenador alternativo	2	3
Alguna baja	Echarle mas horas e intentar acabar lo maximo posible	1	4
Que icaro no funcione	Intentar buscar una solucion con los desarrolladores de icaro o usar otra infraestructura	2	4
Problemas con la base de datos	Hablar con el profesor de BBDD	3	2
Perdida de datos	Procurar tener varias copias	1	3-4
Problemas con Java	Hablar con algun profesor que entienda del tema	2	2
Problemas con la libreria grafica (SWT)	Buscar opciones altenativas	4	2
Migrar a otra version de Icaro	Volver a version anterior	2	1
Problemas interfaz web en Icaro Mini	Intentar buscar una solucion con los desarrolladores de icaro	3	4

Tabla 1. Riesgos

LEYENDA:

Probabilidad:

- 1: Muy improbable
- 2: Poco probable
- 3: Moderadamente probable
- 4: Probable
- 5: Muy probable

Consecuencias:

- 1: No supone ningún problema
- 2: Supone un pequeño problema
- 3: Supone un problema moderado
- 4: Supone un gran problema
- 5: Catastrófico

Diseño UML

Para proporcionar una idea gráfica de cada escenario, se realizaron diagramas de secuencia representativos de los casos de uso. Esto resultó de gran utilidad para comprender mejor el uso de un sistema multiagente y comprobar qué agentes y recursos eran necesarios en cada caso y cuales sobraban.

Además, aportó una idea un poco más clara de las ventanas que era necesario implementar en la interfaz de usuario para cada escenario.

Adjuntamos algunos de ellos como ejemplo.

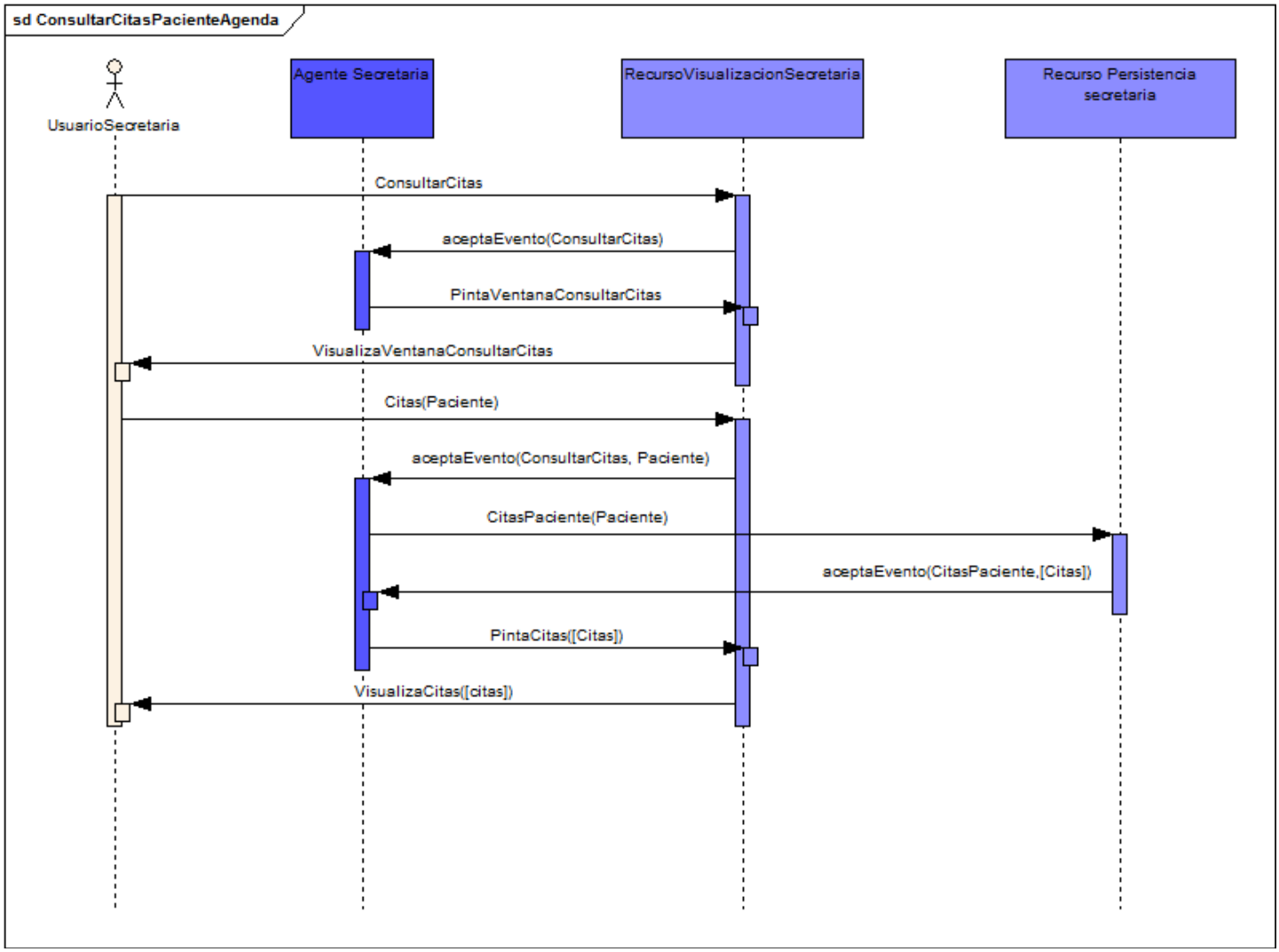

Figura 1. Diagrama Secuencia ConsultarCitasPaciente.

Este diagrama representa a un usuario secretaria, consultando las citas de un determinado paciente. Como se puede ver, para llevar a cabo esta acción, se necesitaría

un agente asociado al usuario y unos recursos con las operaciones aquí descritas implementadas.

A partir del diagrama, se puede intuir el tipo de ventanas que se requiere que se le muestre al usuario durante esta acción, así como la comunicación que es necesaria establecer entre el agente y sus recursos.

Nótese que los recursos no hacen más que cumplir "órdenes" asignadas por el agente. Y que estos solo pueden comunicarse con el agente a través de eventos.

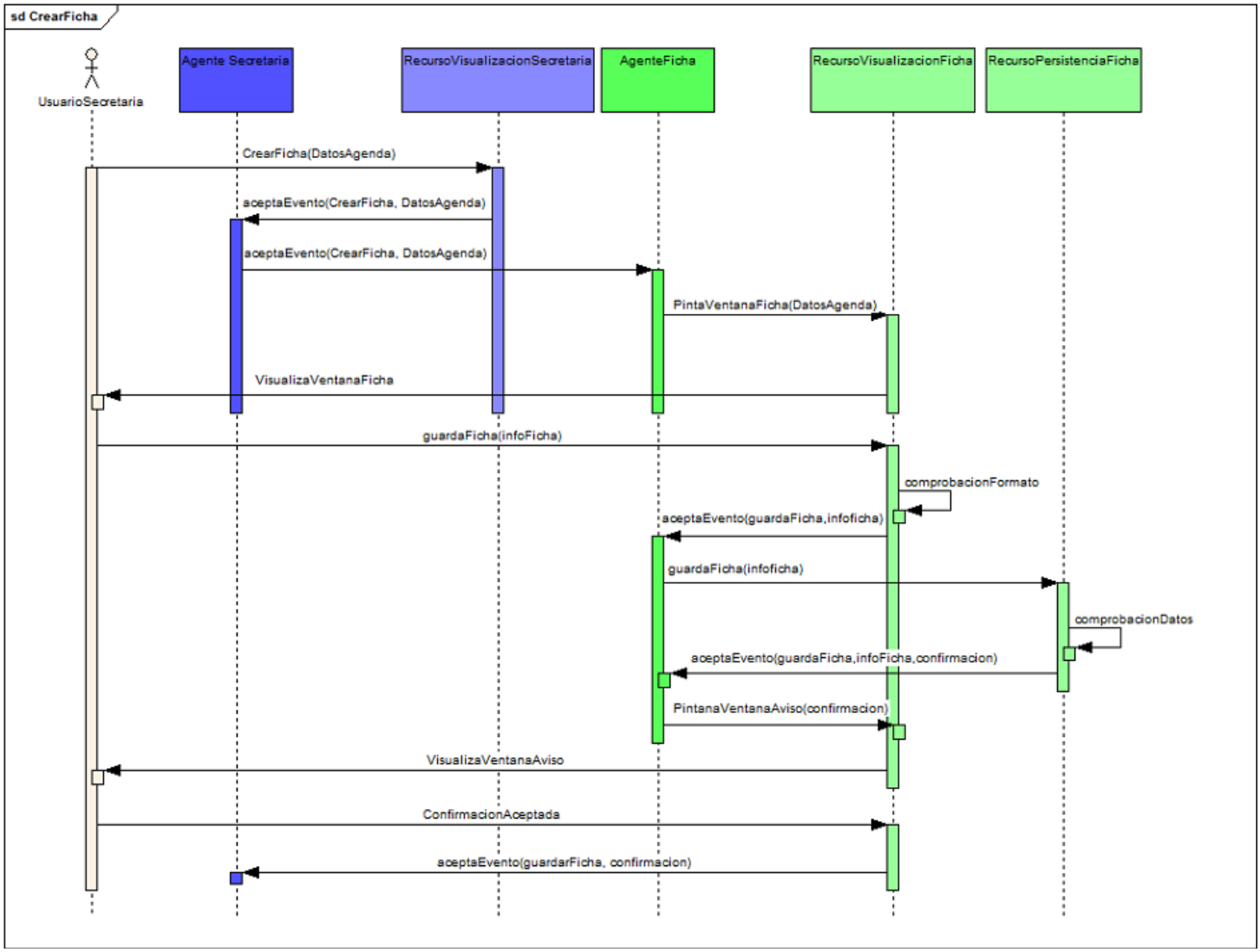

Figura 2. Diagrama Secuencia CrearFicha.

Este diagrama es un poco más complejo que el anterior. Representa a un usuario Secretaria que desea crear una nueva ficha de paciente.

Para poder realizar esta acción, aparte de tener el agente asociado al usuario, como en el caso anterior, es imprescindible comunicarse con otro agente (Ficha). Es el agente Ficha el que tiene asociados los recursos de visualización y persistencia de las fichas de pacientes. Por tanto también todas sus operaciones.

Así pues el agente Secretaria, al recibir la "tarea" del usuario, se la comunica al agente Ficha y este se pone a trabajar. Posteriormente, es el agente ficha, el que da la orden de

mostrar la ventana de nueva Ficha y a continuación el que se encarga de que los datos se almacenen correctamente.

Cuando el agente Ficha comprueba que ha terminado le envía el aviso al agente Secretaria para que dé por terminada la tarea.

Nótese que en este caso, los agentes han colaborado entre sí para ejecutar una tarea, y que para comunicarse entre agentes, es estrictamente necesario el uso de eventos.

Diseño de la interfaz

- Planteamiento general del diseño:

El diseño de la interfaz de usuario se ha basado en los diagramas de secuencia realizados. Estos permiten saber qué ventanas se necesitan en cada momento y qué salida deben mostrar. A continuación se muestra un ejemplo:

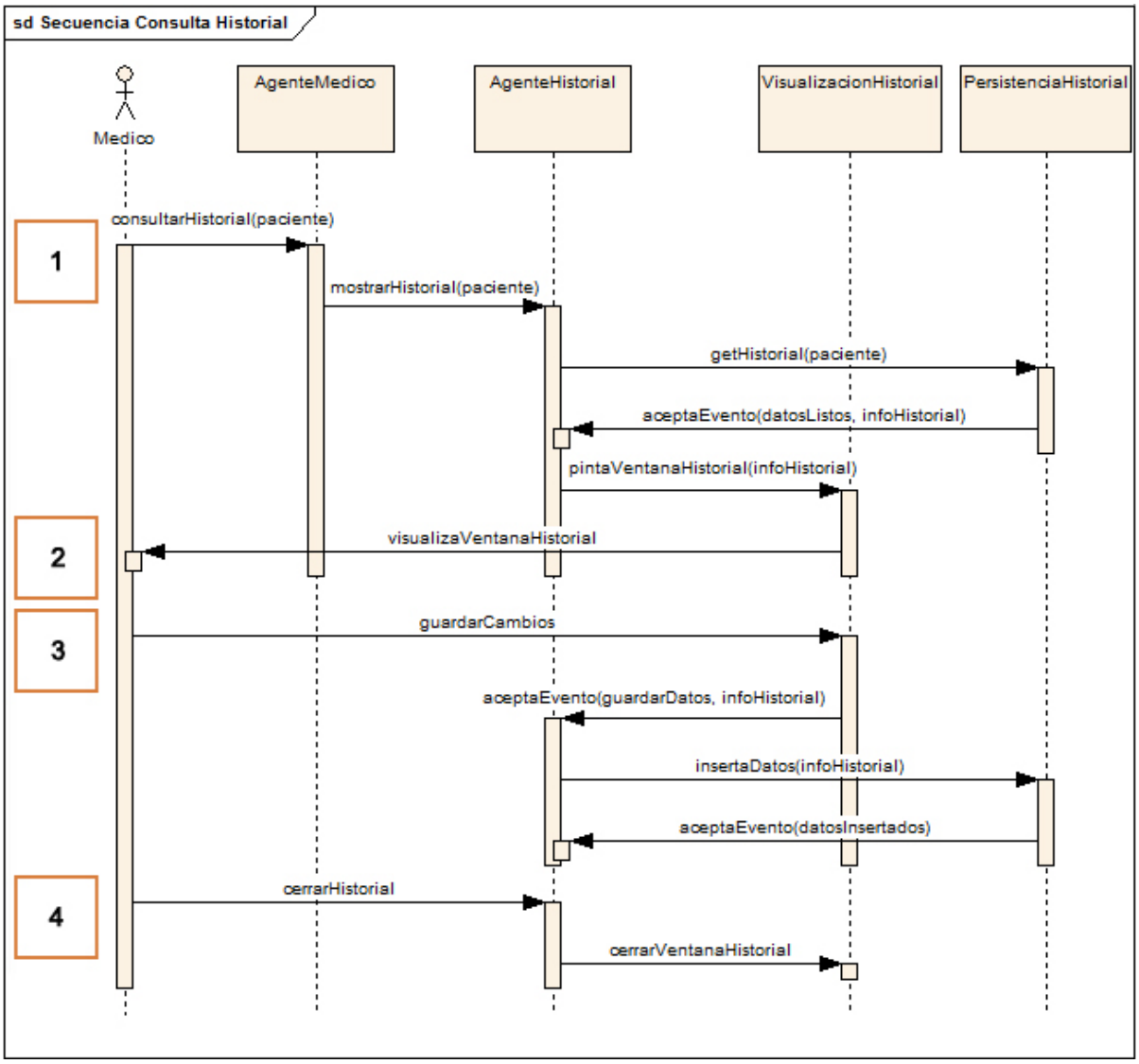

Figura 3. Diagrama de secuencia para sacar interfaces.

Este ejemplo muestra un diagrama de secuencia que representa la acción "Consultar Historial". Tal como se puede ver, de aquí se pueden sacar cuatro interfaces. Tres de ellas de salida y una de entrada.

La interfaz 1 es la ventana que se le muestra al médico en donde se da la posibilidad de

dar a algún botón que active la acción "Consultar Paciente". La interfaz 2 es la ventana donde se le muestra el historial al médico. La interfaz 3 indica que en la ventana de historial se debe permitir al médico ejecutar la acción "Guardar Cambios". Finalmente la interfaz 4, muestra que la ventana de historial debe tener algún botón para ejecutar la acción "Cerrar Historial".

Una vez obtenidos todos los distintos interfaces necesarios, con sus respectivos datos de entrada y de salida, el siguiente paso es realizar un diseño de cada uno de manera que sean apropiados para el tipo de usuario final de la aplicación.

- Consideraciones de usabilidad

No basta con decidir qué interfaces son necesarios. También es muy importante decidir cómo se van a hacer estos interfaces, para que el usuario final tenga una buena experiencia de uso. A esto es a lo que llamamos "Usabilidad".

Durante el diseño de los interfaces se ha tenido en cuenta los siguientes puntos:

- Intuitividad:

Los interfaces deben ser intuitivos para el usuario final. Para ello hay que tener en cuenta el perfil de usuario. Va a ser un usuario con pocos conocimientos en informática por lo que no se puede presuponer que sabrá donde están las cosas o cómo buscarlas. Tampoco sabrá de combinaciones de teclas ni de palabras técnicas como por ejemplo "tabular" o "menú contextual".

Por todas estas razones se ha optado por un diseño más visual. Todo lo que se puede hacer debe estar a la vista y debe poder usarse con solo 2 posibles opciones: Clic o Doble Clic. Se ha prescindido de los menús superiores clásicos de Windows ya que se considera que esto confunde al usuario buscando entre todos los menús la opción adecuada.

Además, se ha intentado usar un texto menos "informático" para las distintas opciones. Por ejemplo, en vez de "Aceptar y Cancelar" se puede usar "Guardar y Cerrar".

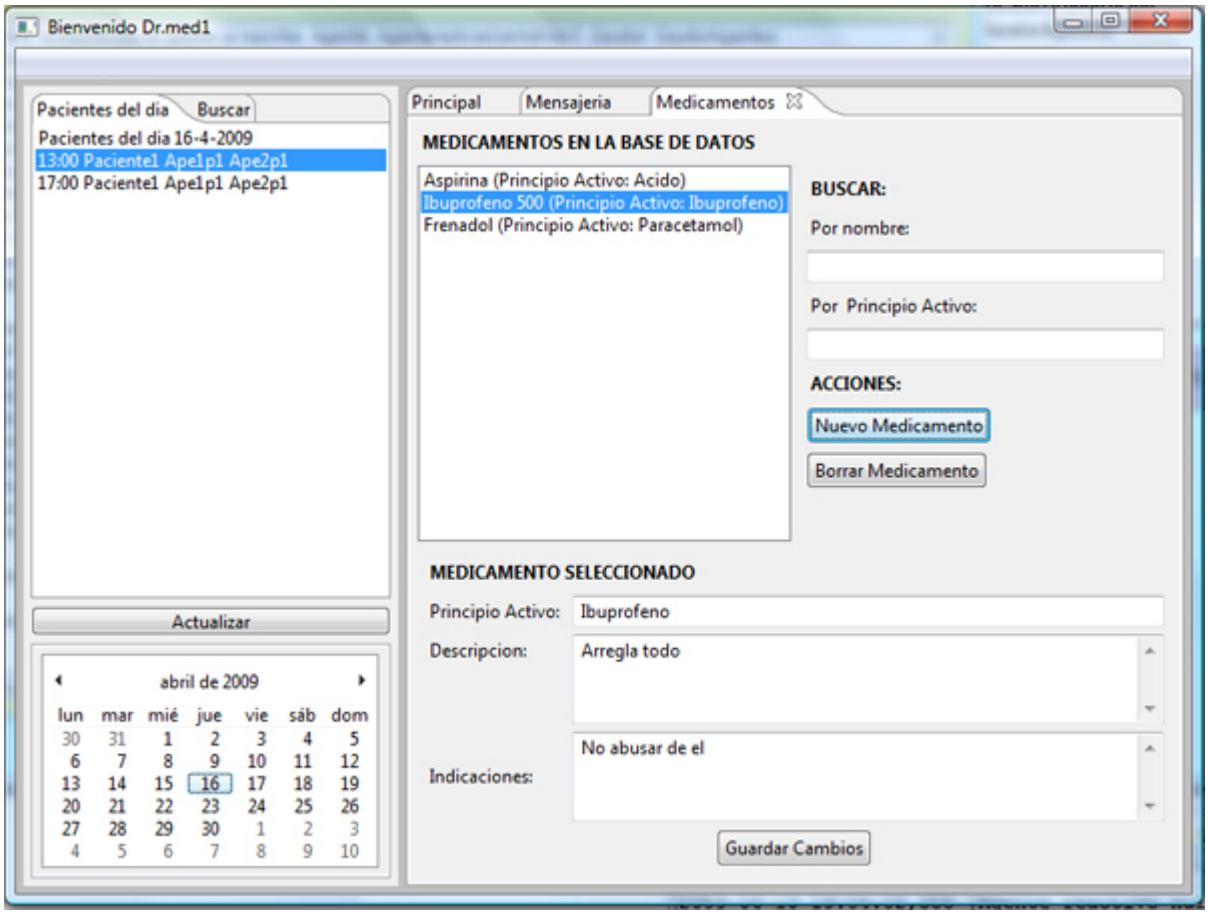

Figura 4. Ejemplo de interfaz 1.

En la imagen se puede ver como todas las opciones están como mucho a dos clicks de distancia, por ejemplo:

- Ver la ficha de un paciente: 1 doble clic.
- Atender a un paciente: Cambiar a pestaña principal y hacer clic en el botón atender.
- Nuevo medicamento: 1 Clic en el botón nuevo medicamento.
- Buscar un paciente: 1 clic para cambiar a la pestaña Buscar.

- Sencillez:

Un médico necesita una aplicación que le permita trabajar de manera fácil y eficiente. No puede permitirse el lujo de perder tiempo intentando averiguar cómo usar las distintas opciones del software. Tampoco le sirve de nada aprender a usar el software, si para cada acción pierde mucho tiempo cambiando de un sitio a otro.

Por estas razones se ha intentado agrupar las cosas según su relevancia. Así el medico no tiene que estar de un lado para otro. Con esto se corría el riesgo de sobrecargar la pantalla con muchas cosas, lo cual iría en contra del concepto de sencillez. Para evitar esto se ha procurado usar pestañas y ventanas auxiliares (por ejemplo, ventanas de búsqueda de medicamentos).

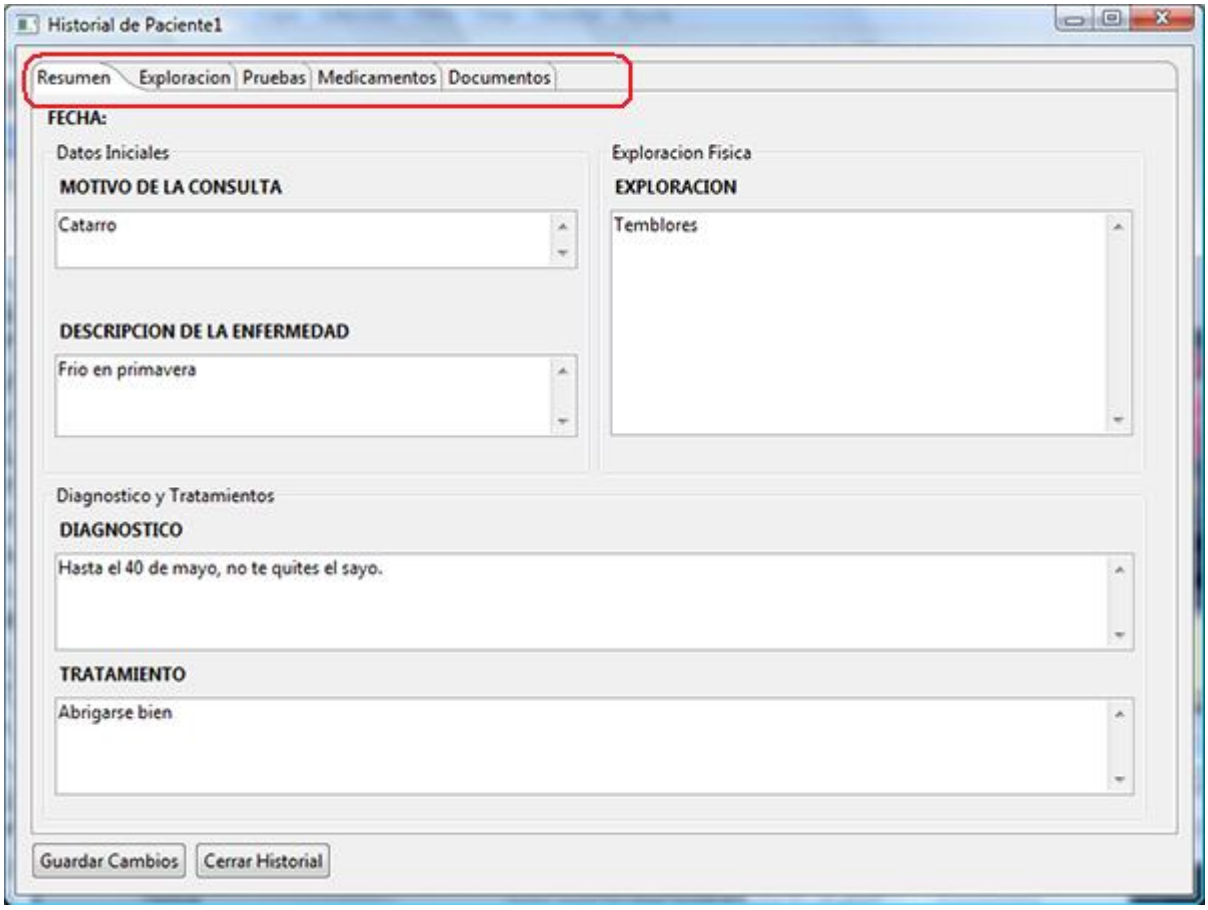

Figura 5. Ejemplo de interfaz 2.

En la imagen se ha remarcado el uso de pestañas. De esta manera en una sola ventana se puede agrupar toda la funcionalidad relevante sin sobrecargar al usuario.

- Mensajes y tratamiento visual de errores

Esta parte también es muy importante a la hora de desarrollar una interfaz de usuario. Igual que hay que hacer intuitivas todas las opciones que ofrece la interfaz, también hay que hacer intuitivos los errores.

Se ha hecho mucho hincapié en capturar todos los errores directamente en la interfaz para así evitar entradas erróneas al sistema, que más adelante podrían causar problemas mayores. Esto implica que al usuario le saldrán ventanas de error cada vez que introduzca datos con formato erróneo. Además, se ha intentado que los mensajes de error sean lo mas claros posibles para que el usuario entienda fácilmente qué es lo que ha hecho mal y cómo puede solucionarlo.

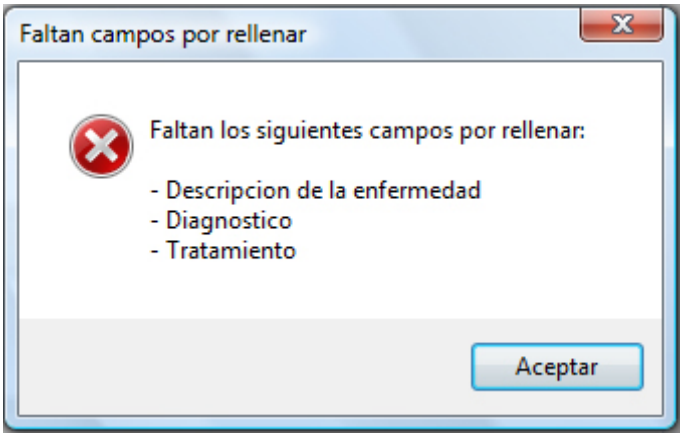

Figura 6. Ejemplo de interfaz 3.

En la imagen se ve un ejemplo de lo que pasa cuando un usuario de tipo médico intenta guardar un historial incompleto. El agente "Historial" le avisa mediante un mensaje de error y el lenguaje usado es lo menos técnico posible.

Diseño de la BD

Para la Base de datos del sistema se barajaron varias posibilidades. En principio, se pensó en usar una base de datos local y otra remota que se fueran sincronizando periódicamente, para así evitar que la aplicación siga funcionando en caso de perder la conexión con la base de datos remota.

Finalmente, se optó por usar solo una base de datos que puede ser tanto local como remota, ya que la localización de la base de datos es indiferente para el buen funcionamiento del sistema.

Para realizar un diseño correcto de la base de datos, se uso el esquema de entidad relación, junto a otras metodologías de diseño aprendidas en la asignatura de bases de datos. El diseño entidad relación consiste en ver como interactúan entre sí las distintas entidades del sistema y definir qué atributos tiene cada una.

Lo primero fue definir las entidades y a continuación realizar el diagrama entidad relación. Para el diagrama, se usó un programa específico de realización de diagramas entidad-relación que hicieron unos alumnos de la UCM como proyecto de fin de carrera en años anteriores.

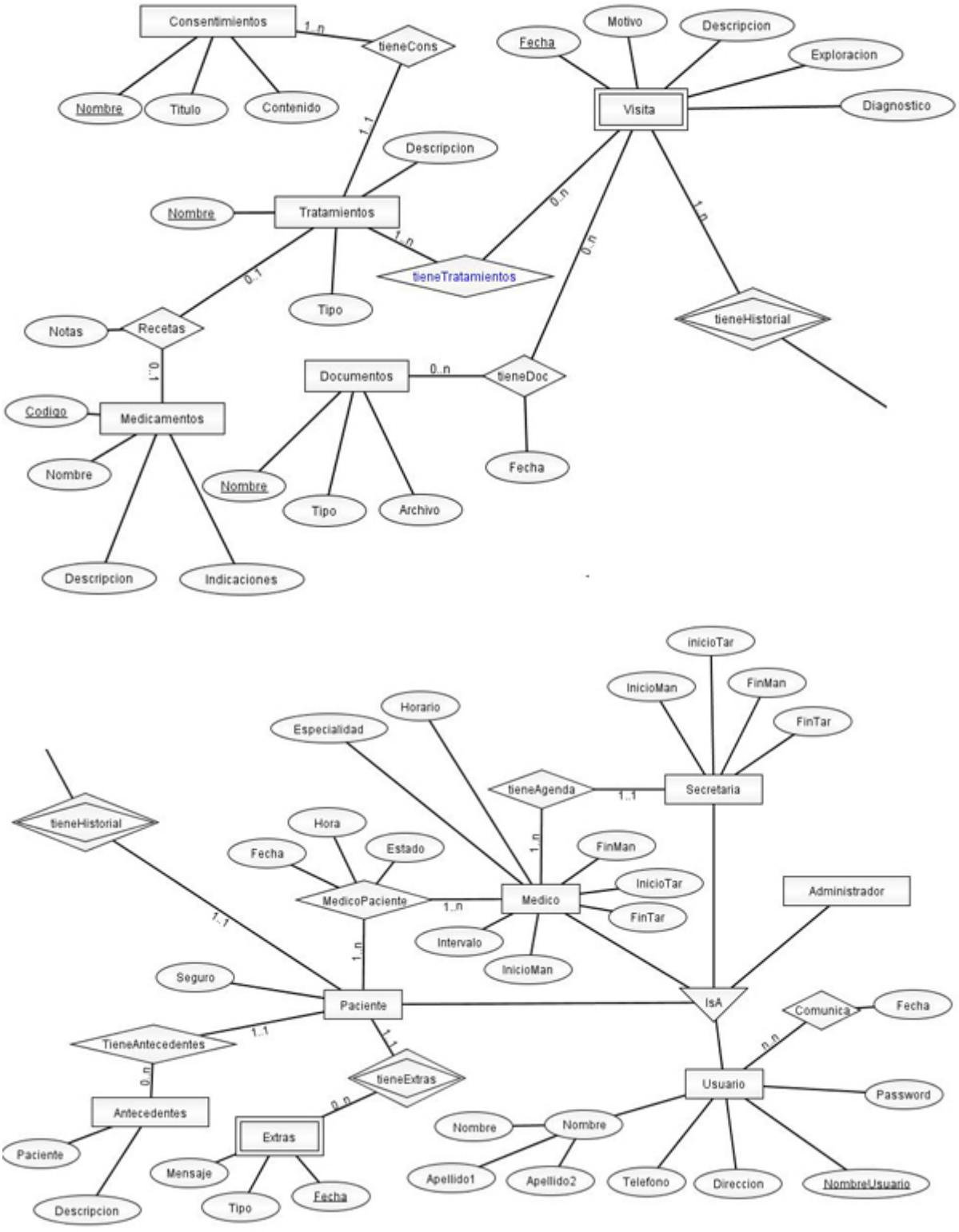

Figura 7. Diagrama Entidad-relación de la base de datos

En la figura se puede ver el diagrama entidad relación. Las entidades vienen representadas por un cuadrado, las relaciones por un rombo y los atributos por un círculo.

Tras esto, hay que transformar el diseño entidad-relación en un modelo relacional. Esto consiste en obtener las tablas finales como consecuencia de una simplificación del diagrama entidad-relación. Con esto se consigue que no se repita la misma información en distintas tablas y que cada tabla sea lo mas sencilla posible.

Arquitectura

Antes de continuar, vamos a introducir el concepto de sistema multiagente. Un sistema multiagente (SMA) está formado por:

1. Un entorno.

2. Un conjunto de objetos.

Se encuentran relacionados con el entorno

- Son pasivos, pueden ser percibidos, creados, destruidos y modificados por agentes.

3. Un conjunto de agentes.

- Son objetos especiales que representan las entidades activas del sistema.
- Tienen habilidad social (Interacción, delegación, coordinación, cooperación y negociación)

4. Un conjunto de relaciones que unen objetos y agentes.

5. Un conjunto de operaciones

- Hacen posible que los agentes perciban, produzcan, consuman, transformen y manipulen objetos.

6. Operadores que representan la aplicación de operaciones sobre el mundo y la reacción de éste al ser alterado.

- Estos operadores se pueden entender como las leyes del universo.

Veamos ahora un ejemplo de como funciona un SMA ambientado en nuestra aplicación. Escenario: El médico quiere consultar sus mails desde la interfaz de usuario.

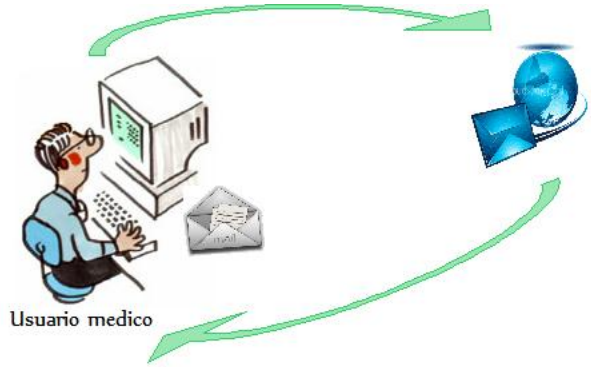

Figura 8. Concepto SMA

El agente (médico) monitoriza la actividad del usuario:

- "Lee/escucha" al usuario.
- Reconoce las "órdenes" del usuario.

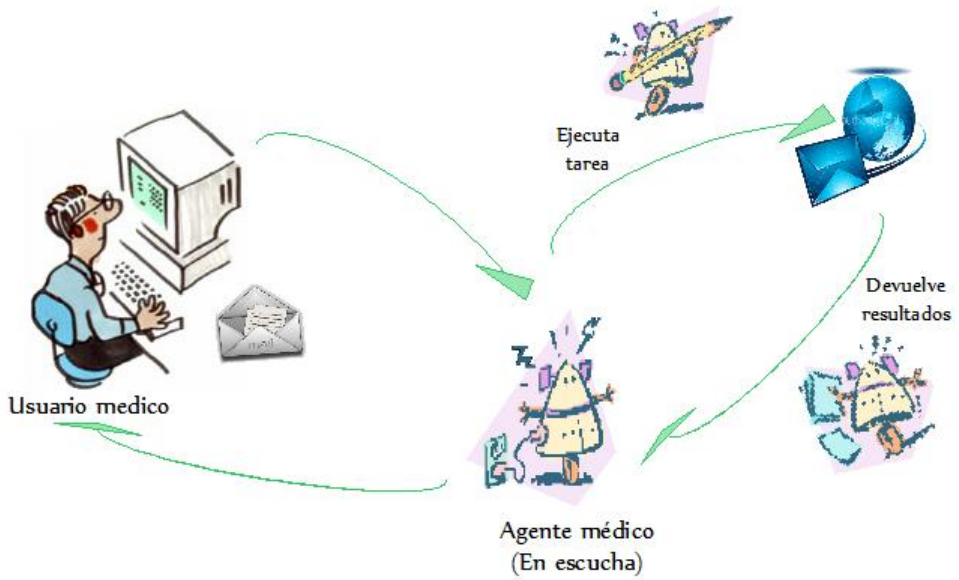

Figura 9. Concepto SMA. Monitorización Agente.

Mientras el usuario está usando la interfaz (recurso de visualización del agente medico), el agente está a la espera de recibir información que le indique que debe realizar alguna tarea para el usuario.

2. El agente persigue satisfacer sus objetivos

- Toma decisiones.
- Puede descomponer objetivos en subobjetivos.
- Ejecuta tareas.

Figura 10. Concepto SMA. Acciones Agente

Cuando el agente recibe una petición (en este caso, consultar mensajería), la analiza y ejecuta las acciones pertinentes para satisfacerla de forma transparente al usuario.

3. Para cumplir objetivos necesita la colaboración con otros agentes (mensajería)

- Negociación
- Delegación
- Coordinación

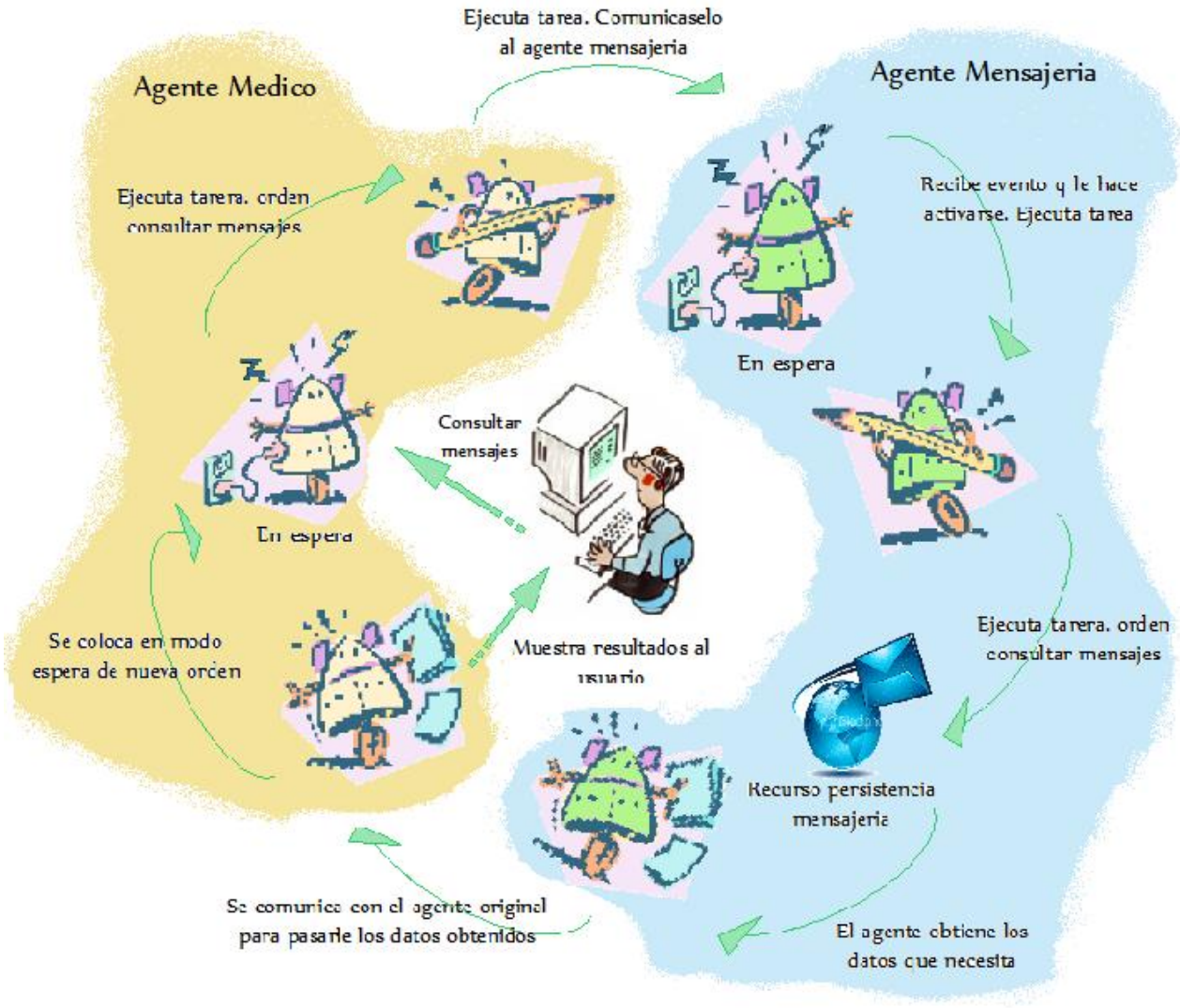

Figura 11. Concepto SMA. Ejecución tareas, comunicación entre agentes.

Para realizar algunas tareas, el agente puede necesitar comunicarse con otros agentes, pedirles información o hacer que trabajen juntos. En este caso, el agente Médico se da cuenta de que esta tarea le concierne en su mayor parte al agente mensajería.

Como resultado le manda un evento avisándole. Este se activa y realiza las tareas que tenga asignadas a dicha petición, devolviendo el resultado al agente que se lo solicitó o comunicándose él mismo con el usuario.

4. Los agentes se comunican con sus propios recursos. Visualización (interfaz de usuario) y persistencia (acceso a los datos de la bbdd)

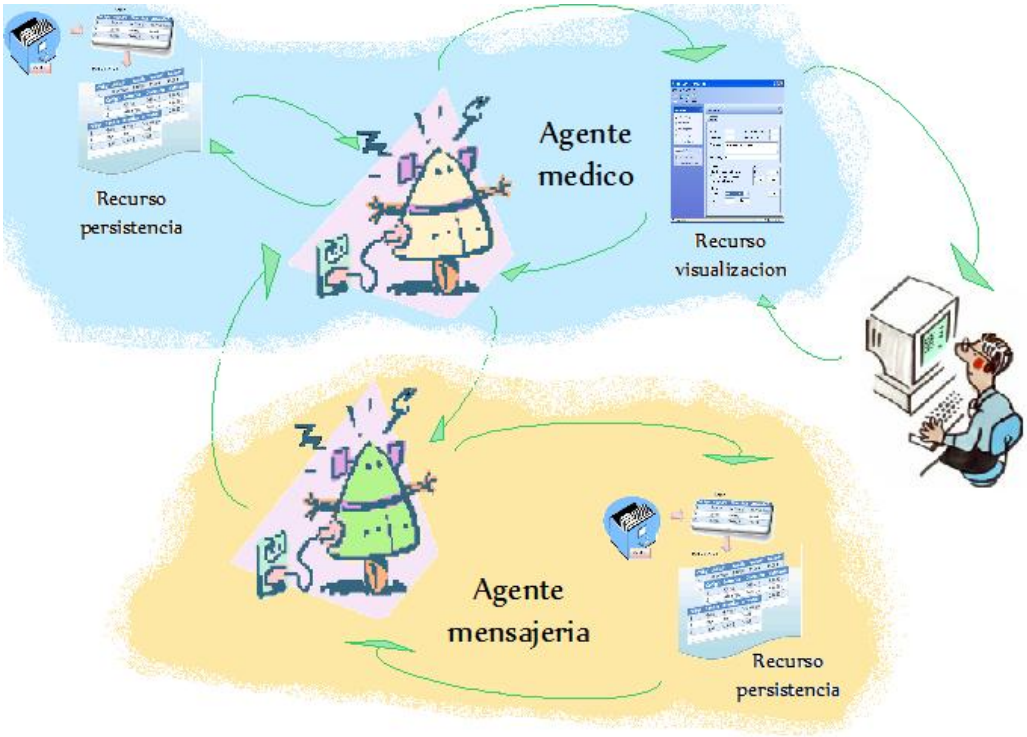

Figura 12. Concepto SMA. Agentes y sus recursos.

Como se puede ver en la imagen, cada agente tiene sus propios recursos en los que no interfieren otros agentes. Esto permite una implementación muy modulable.

5. El resultado de la consulta es devuelta al usuario a través de su agente (médico)

En el caso de Icaro, se parte del modelo de vocales de Yves Demazeau:

O: Un patrón de Organización donde hay
A: Agentes con roles Gestor y Agente Aplicación
E: Entorno formado por Recursos
I: Modelo de Información donde se meten las clases del dominio de aplicación
I: El modelo dinámico. Son la Interacciones entre agentes y recursos para llevar a cabo los objetivos o los requisitos de la aplicación.
U: El uso o la utilidad de la aplicación.

En general la metodología que se ha seguido es:

En primer lugar, la de asignar un agente a cada rol de la aplicación definiendo sus propios recursos.

En segundo, asignar agentes a tareas específicas que comparten recursos de manera que se divida la funcionalidad en pequeños componentes fácilmente tratables de manera individual.

A continuación se muestra en detalle.

Arquitectura de la aplicación.

Inicialmente se planteó la idea de colocar un agente por cada rol del sistema. Dividiendo la funcionalidad de la aplicación en 4 grandes partes (médico, secretaria, paciente, administrador).

Posteriormente se asignaron agentes a labores específicas como son las tareas relativas a ficha, historial, medicamentos, mensajería o login. Estos últimos son activados por alguno de los agentes asignados a un rol, excepto login que se activa en el arranque del sistema.

En general la división en agentes se aplicó en función del conjunto de operaciones que afectan a una estructura de datos común. Por ejemplo la ficha como estructura de datos contiene información personal relativa al paciente.

Se creó un agente con dicho nombre porque asociada a una estructura de tipo ficha, se debían implementar varias operaciones (mostrar una ficha por ventana, crear, modificar o eliminar, etc.). A todas aquellas estructuras de datos que requerían un conjunto de operaciones bien definido, se le asignó un agente y unos recursos propios.

A continuación adjuntamos el estudio que se aplicó para determinar qué agentes debía tener la aplicación:

Estudio de agentes:

Las operaciones que aparecen asignadas a cada agente vienen de los casos de uso que se han definido con anterioridad.

Agente Acceso: Acciones correspondientes al login (introducir datos, validación, identificación clase de usuario (médico, secretaria, Paciente, administrador)).

Comunica con los agentes Médico, Secretaria, Paciente, Administrador

AgenteSecretaria: Acciones propias del menú secretaria.

Operaciones:

- Crear, consultar, modificar ficha.
- Establecer estado de un paciente.
- Buscar visitas por paciente.
- Tiempo estimado de entrada.
- Dar cita.
- Enviar, consultar, eliminar mensaje.
- Ver próximas citas.
- Consultar estado consulta.
- Comunica con los agentes Ficha, Mensajería, Acceso.

AgenteMedico: Acciones propias del menú del médico.

Operaciones:

- Modificar, consultar historial de un paciente.
- Crear Visita.
- Crear, modificar, eliminar, imprimir receta de un paciente.
- Crear, modificar, eliminar, imprimir consentimiento.

- Enviar, consultar, eliminar mensaje.
- Consultar Agenda.
- Añadir, consultar, eliminar documentos de un paciente.
- Crear, modificar, eliminar, consultar medicamento.
- Añadir documentos.
- Comunicación con los agentes Historial, Mensajería., Medicamentos, Acceso.

AgentePaciente: Acciones propias del menú del paciente.

Operaciones:

- Ver próximas citas.
- Consultar estado consulta.
- Enviar, consultar, eliminar mensaje.
- Consultar, modificar ficha.
- Consultar receta.
- Subir Documentos.
- Comunicación con los agentes Mensajería, Agenda, Historial, Acceso.

Nota: El agente Paciente no está implementado puesto que se usa únicamente en la interfaz web. Después de algunos intentos fallidos de diseño de dicha interfaz se propone exclusivamente como versión futura.

AgenteAdmistrador: Acciones propias del menú Administrador.

Operaciones:

- Configuración de la aplicación.
- Añadir, modificar, eliminar usuario.
- Comunicación con los agentes Ficha, Acceso.

AgenteFicha: Acciones sobre las fichas.

Operaciones:

- Crear, consultar, modificar ficha.
- Comunicación con los agentes Secretaria, Médico.

AgenteMensajeria: Acciones sobre los mensajes.

Operaciones:

- Enviar, consultar, eliminar mensaje.
- Comunicación con los agentes Secretaria, Paciente, Médico, Administrador.

AgenteHistorial: Acciones sobre el historial de un paciente.

Operaciones:

- Modificar, consultar historial.
- Modificar, eliminar, imprimir receta.
- Modificar, eliminar, imprimir consentimiento.
- Consultar, eliminar documentos.

- Comunicación con los agentes Paciente, Médico.

AgenteMedicamentos: Acciones sobre los medicamentos.

Operaciones:

- Añadir, consultar, eliminar medicamento.
- Comunicación con los agentes Médico.

NOTA: Cada agente tiene su recurso de visualización y persistencia.

La arquitectura resultante es un sistema multiagente que consta de los siguientes agentes y recursos distribuidos por colores:

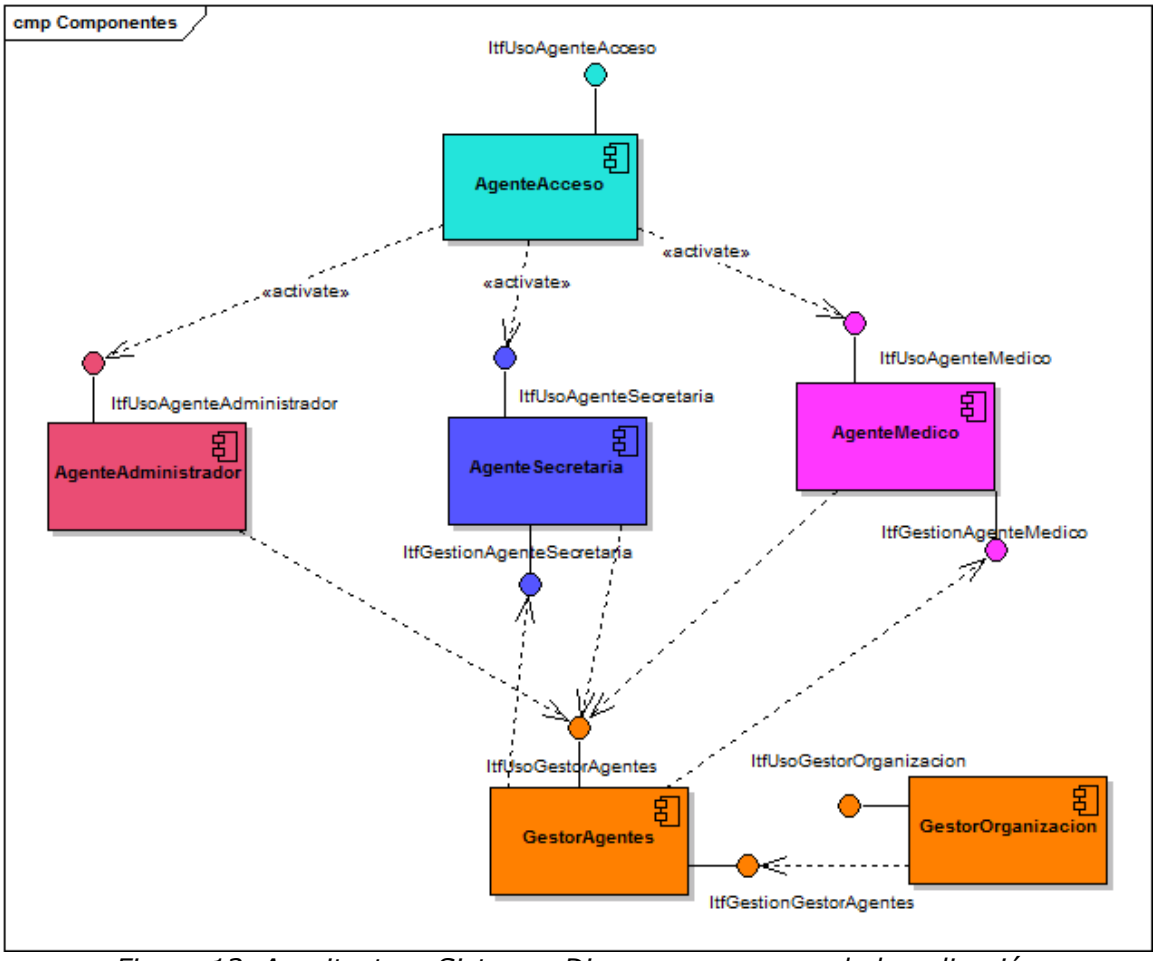

Figura 13. Arquitectura Sistema. Diagrama arranque de la aplicación.

Según el usuario con que se acceda a la aplicación el agente login activará alguno de los agentes asignados a un rol.

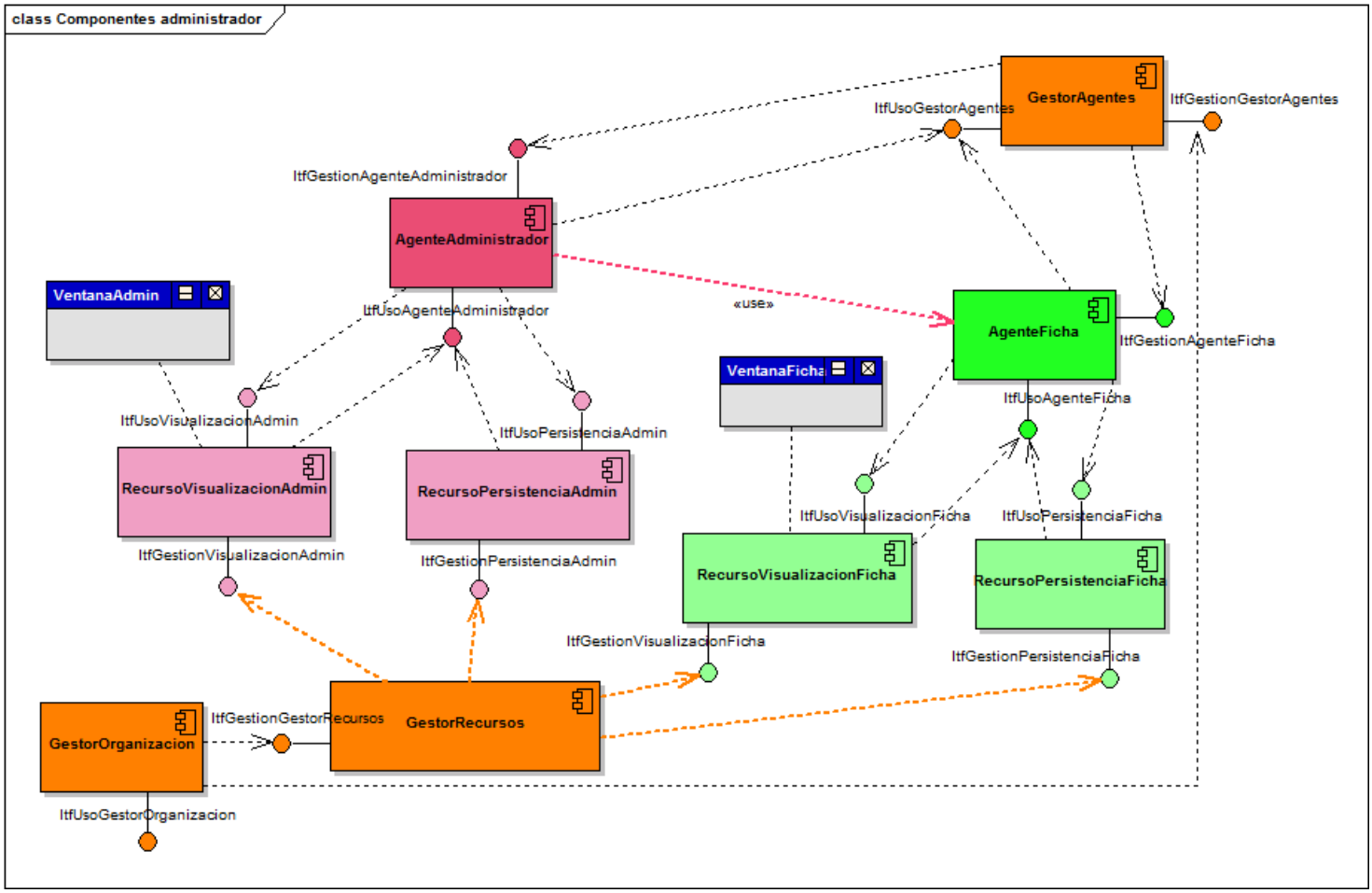

Figura 14. Arquitectura Sistema. Agente administrador.

Muestra los agentes con los que se comunica y recursos que tiene asignados el Agente administrador.

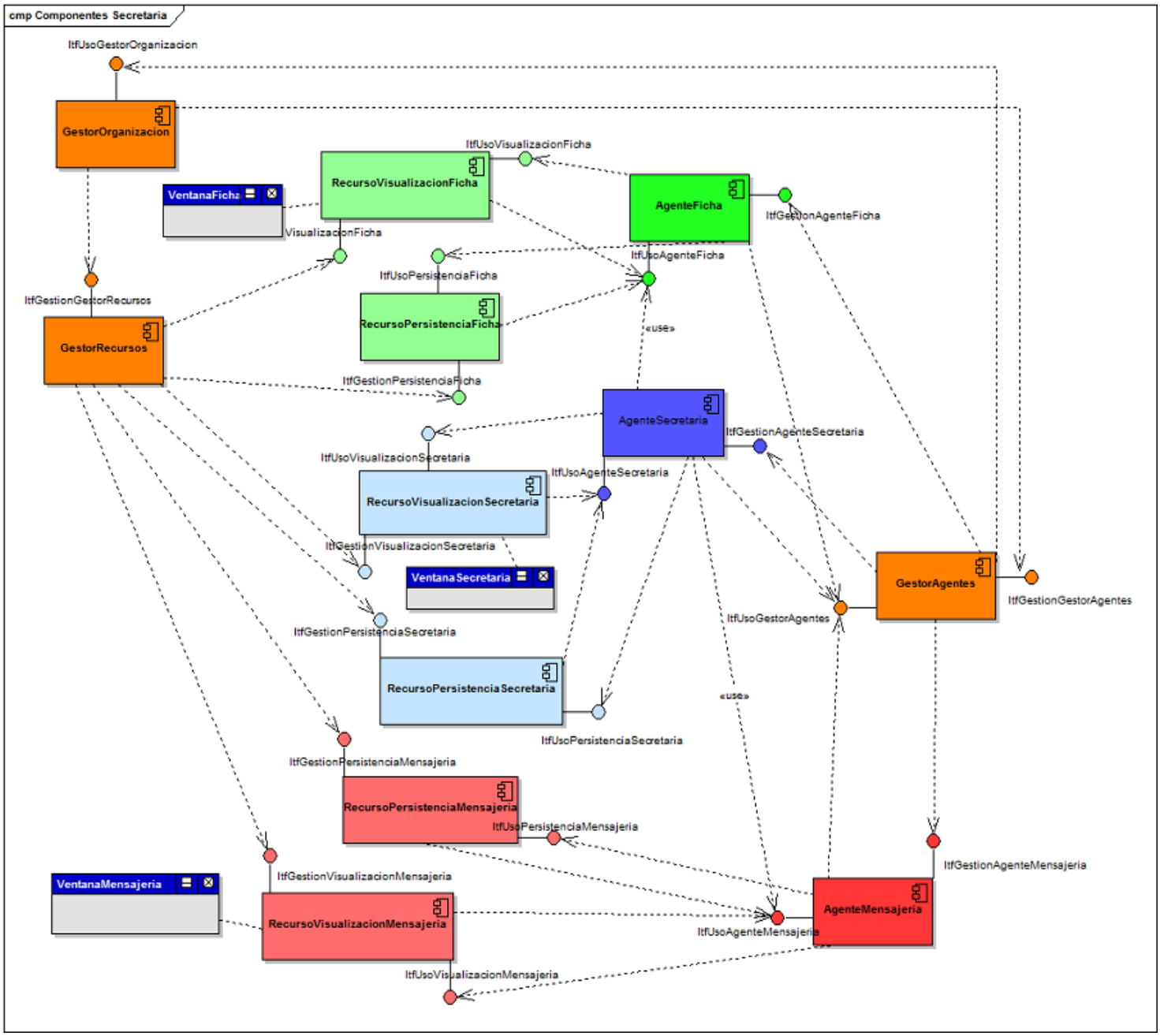

Figura 15. Arquitectura Sistema. Agente Secretaria.

En este diagrama se detallan los agentes con los que se comunica y recursos que tiene asociados el agente Secretaria.

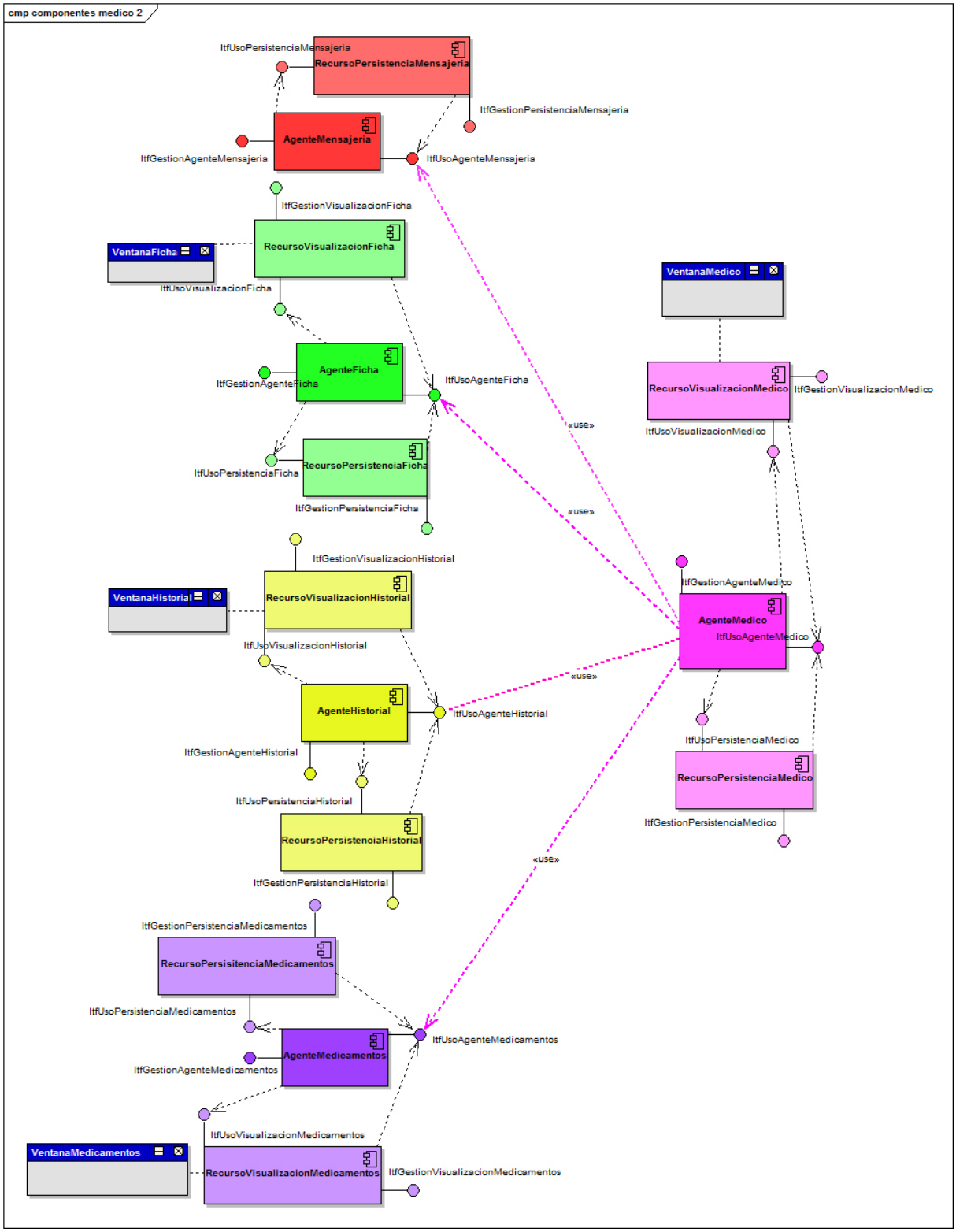

Figura 16. Arquitectura Sistema. Agente Médico.

Este diagrama muestra los agentes con los que se comunica y recursos que tiene asignados le agente Medico.

Leyenda:

- Los componentes en color naranja pertenecen a la plataforma ICARO-T.
- Los agentes tienen cada uno un color.
- Los recursos tienen un color ligeramente más claro que el del agente al que están asociados.
- A los recursos de visualización se les ha adjuntado la ventana más significativa que implementa, que no tiene porque ser la única.

Como se puede ver en las imágenes, cada uno de los roles del sistema tiene su propio agente (Secretaria, Médico y Administrador) y estos a su vez tienen sus propios recursos de visualización y persistencia.

Durante el diseño de la aplicación se planteó la idea de mantener un recurso de persistencia centralizado para todos los agentes. Al final para mantener el grado de modularidad inicial se decidió mantener un recurso de persistencia por agente igual que en la visualización.

Arquitectura ICARO

Igual de importante que la arquitectura de la aplicación es la estructura de la organización de la plataforma que la gestiona. Vamos a explicar brevemente su diseño.

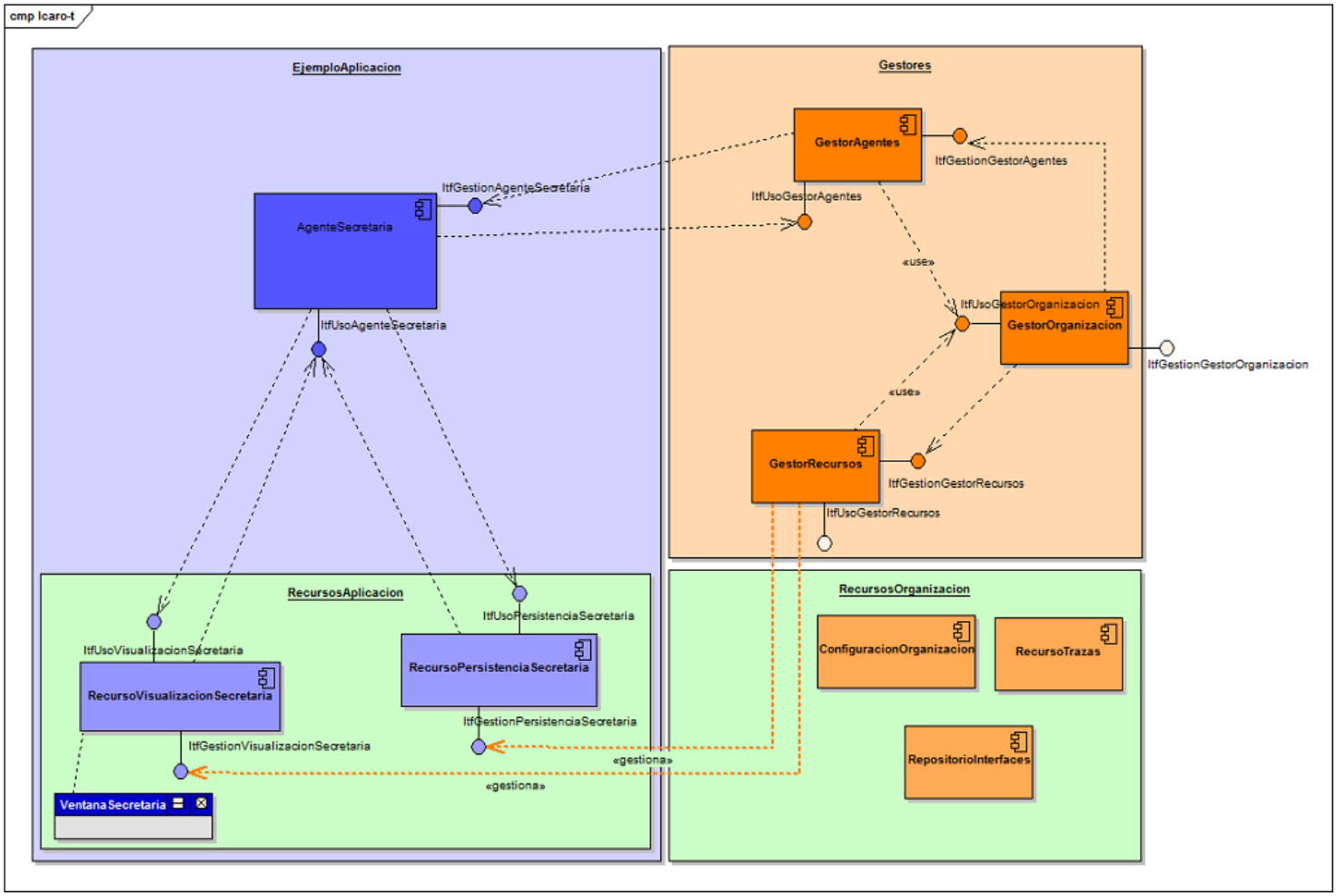

Figura 17. Arquitectura ICARO

En la parte izquierda de la figura se encuentran los componentes que implementan la funcionalidad de la aplicación: El agente Secretaria, los recursos de visualización y de persistencia.

En la parte derecha se encuentran los componentes proporcionados por ICARO:

- **Los gestores**. Se encargan de la creación, arranque, parada y monitorización de los componentes de la aplicación. El gestor de organización crea y monitoriza a los gestores de agentes y de recursos y éstos crean y monitorizan los agentes y recursos de la aplicación.
- **Los recursos de la organización**. Permiten la creación de la infraestructura inicial de la organización, incluidos los gestores, y que dan servicio tanto a los gestores como a los agentes y a los recursos de la aplicación.
    - **Configuración de la organización y Repositorio de interfaces.** Agentes, Recursos y Modelo de información constituyen los elementos básicos para construir aplicaciones, pero para que la aplicación funcione utilizando ICARO es necesario definir explícitamente una Organización. La organización que implementa el sistema de acceso se define con un documento formal –expresado en XML que detallaremos más adelante- donde se especifican las propiedades y los componentes de la organización, los roles, atributos, características y dependencias de los componentes. Este documento se utiliza para crear la organización, es decir, para crear los componentes computacionales que deben llevar a cabo la funcionalidad del sistema.
    - **Recurso Trazas**. Permite realizar un seguimiento de trazas a lo largo de la ejecución de las aplicaciones.

La comunicación entre los componentes de la arquitectura, se lleva a cabo a través de interfaces, de forma que la implementación interna queda oculta a los potenciales clientes de cada componente. En la figura cada componente tiene una interfaz de uso y otra de gestión.

Implementación

Pautas de Codificación

Para mantener un orden y permitir una comprensión clara del código se han seguido unas pautas de codificación. Este aspecto es muy importante ya que facilita el trabajo en equipo. No basta con dividir las tareas entre los integrantes del grupo sino que hay que llegar a un consenso sobre cuestiones de documentación, estructura y nombrado.

Varias de las pautas establecidas se han heredado de ICARO-T con el fin de hacer el código lo mas homogéneo posible a la infraestructura. De esta manera, cualquiera que conozca ICARO-T podrá entender rápidamente lo que hace un agente y sus recursos. Otras son pautas ya generalizadas entre los programadores de Java, que aunque no son oficiales se podría decir que son "buenas prácticas".

A continuación están listadas las pautas de codificación establecidas para desarrollar este proyecto, divididas en categorías:

- Documentación:
  - Para la documentación se ha usado JavaDoc. Esta herramienta de Java permite ir haciendo la documentación a la par que el código, lo cual hace que se pueda mantener todo el código documentado de forma sencilla y actualizada.

    El entorno de desarrollo Eclipse facilita el uso de JavaDoc ya que crea una plantilla de la documentación de cada método, con sus parámetros encima de la definición, para que después el desarrollador la rellene.

- Estructura:
  - Se ha estructurado cada agente en paquetes para separar las distintas partes del agente. Esta estructura se ha heredado de los ejemplos de aplicaciones que vienen en ICARO-T y a partir de ahí todos los agentes realizados se han implementado siguiendo la misma estructura para conseguir la homogeneidad. Más abajo se explica en detalle la estructura del código de los agentes.
- Nombrado:
  - Cada una de las ventanas de la interfaz está separada en una clase llamada PanelX.java. Por ejemplo: PanelAgenda.java
  - Las Clases De Dominio usadas para enviar la información entre agentes se llaman InfoX.java. Por ejemplo: InfoCita.java
  - La clase con la que se comunica la interfaz para enviar eventos a los agentes se llama UsoAgenteX.java. Por ejemplo: UsoAgenteHistorial.java
  - En cuanto al uso de SWT, según el tipo de widget usado se declara la variable correspondiente de una manera concreta.
    - Text: tAlgo
    - Label: lAlgo
    - Combo: cAlgo
    - TabFolder: tabAlgo

Organización del código

El código de la aplicación es básicamente un conjunto de agentes que se comunican entre sí. Cada agente es independiente, por lo que se ha definido una estructura estándar que debe tener cada uno de los agentes. Esto hace que la creación de nuevos agentes sea más rápida y que cualquier programador pueda entender fácilmente el funcionamiento de un agente que ha programado otra persona.

Esta estructura se basa en separar las distintas partes del agente en distintos paquetes. Por un lado están los recursos, por otro están las clases de dominio y aparte están las clases del agente en sí. A continuación se puede ver más gráficamente la estructura:

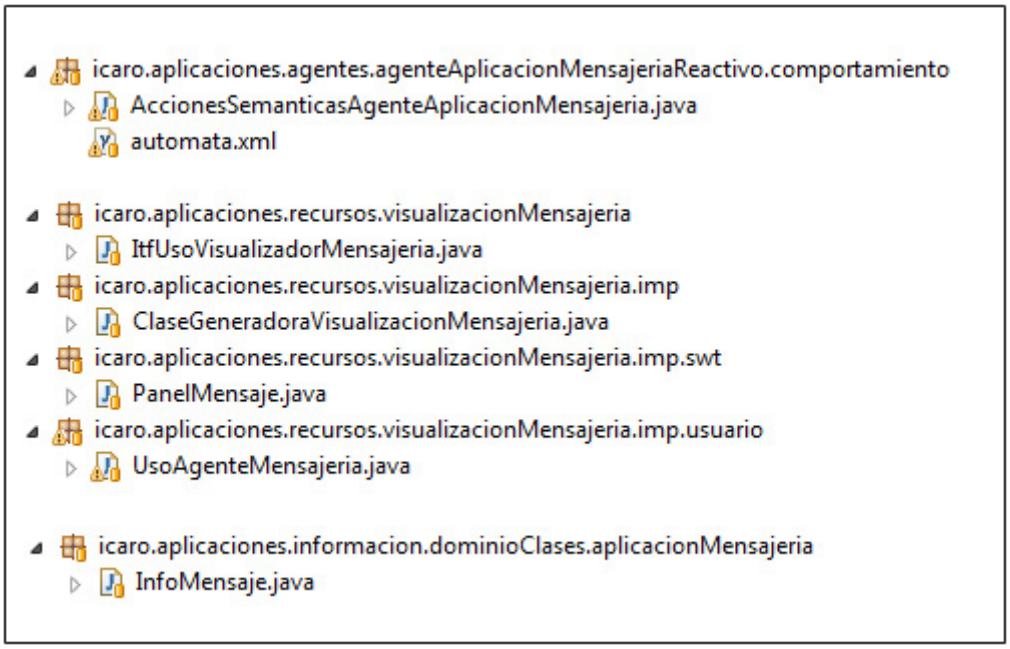

Figura 18. Estructura de las clases de un agente.

Viendo que todos los agentes eran prácticamente iguales en cuanto a clases base se refiere, se tomó la decisión de hacer un "Template" para la creación de agentes. Un "Template" es básicamente una plantilla con los paquetes y clases comunes a todos los agentes. No solo en continente (estructura) sino también en cuanto al contenido, ya que los métodos básicos de las clases mostradas en la figura son todos comunes.

El hecho de usar un template ayudó a ahorrar mucho tiempo a la hora de crear nuevos agentes y evitó cometer errores comunes. Esto está explicado más detalladamente en el apartado de "Resultados y Conclusiones".

Interfaz

La interfaz se ha implementado usando una librería llamada SWT creada por los desarrolladores de Eclipse. Es una librería gráfica basada en widgets, muy versátil, fácil de usar y multiplataforma.

La integración en ICARO-T se ha hecho de la siguiente manera:

- Cada ventana es una clase dentro del recurso de visualización del agente en cuestión.

- SWT requiere un bucle que va comprobando constantemente sí hay que actualizar la ventana. Esto provoca un bloqueo en el thread del agente. Para evitar el bloqueo, cada ventana tiene su propio thread, lo cual se consigue extendiendo a la clase thread.

```
while (!shell.isDisposed()) {
    if (!disp.readAndDispatch())
        disp.sleep();
}
```

Figura 19. Código swt 1.

- Los datos a pintar en la ventana generalmente se obtienen de forma asíncrona. Por lo tanto, primero se pinta la ventana vacía y después se rellena con los datos. El usuario final no se percata de esto ya que suele ser un proceso rápido.

- La comunicación entre la ventana y el agente se hace mediante una clase intermedia llamada "UsoAgenteX". Posteriormente esta clase se ocupa de enviar los eventos a los agentes y de devolver el resultado a la ventana.

- El hecho de tener la ventana en un thread propio hace que la comunicación entre el agente y la ventana sea una comunicación entre threads distintos. Es más, normalmente hay que pasar por varios threads desde que la ventana hace una petición hasta que se devuelven los datos. Esto hace que salten excepciones de tipo "Invalid Thread Access" al intentar comunicar un agente con una ventana. Para evitarlo se ha tenido que usar un "código de seguridad" que consiste en encapsular el código SWT que va a ser llamado desde un thread externo.

```
public void destruir(){
    disp.asyncExec(new Runnable() {
        public void run() {
            shell.dispose();
        }
    });
}
```

Figura 20. Código swt 2.

En la figura se puede ver lo que nosotros denominamos como "código de seguridad". Dentro de *public void run() { ... }* se pone cualquier código que tenga que modificar widgets swt desde otro thread. Todo lo que lo rodea es el código de seguridad.

El funcionamiento de SWT se explica a continuación:

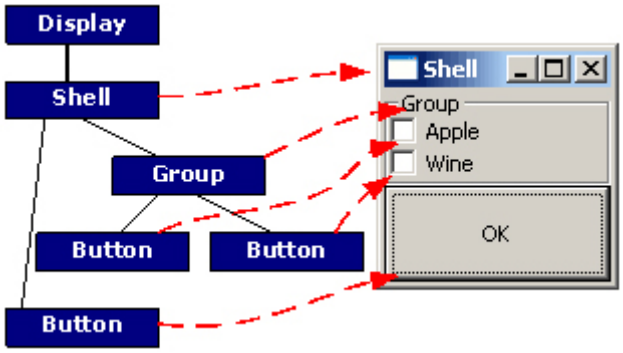

Figura 21. Diagrama funcionamiento SWT.

Cada aplicación tiene un Display que se ocupa de pintarla y actualizarla en todo momento. En nuestro caso, cada agente tiene un Display. Posteriormente se define un Shell por cada una de las ventanas que se crean. Un shell es el "contenedor" de todos los widgets que tiene la ventana.

Dentro de un shell se pueden meter widgets de muchos tipos. Estos son algunos de ellos, los que más se han usado en la aplicación:

- Composite: Es un contenedor que permite agrupar widgets para poder organizarlos localmente. Varios "composites" se organizan dentro de un shell y después otros widgets se organizan dentro de cada composite.
- Group: Igual que un composite pero con la posibilidad de ponerle un título visible al grupo.
- Button: Widget para crear botones a los que posteriormente se le pueden asignar eventos, por ejemplo ClickEvent.
- Label: Widget para crear etiquetas. Sirve para mostrar texto no modificable en pantalla.
- Text: Widget para crear campos de texto. Sirve para mostrar texto modificable en pantalla.

La organización de los widgets dentro de una ventana se hace siguiendo un "layout". Hay 4 tipos de layouts: GridLayout, FormLayout, FillLayout y RowLayout. Para este proyecto se ha optado por usar GridLayout que consiste en dividir un contenedor en una malla de 1 o más columnas. Los widgets que se crean se van mostrando en pantalla primero llenando las columnas definidas y una vez usadas todas las columnas se crea una nueva fila.

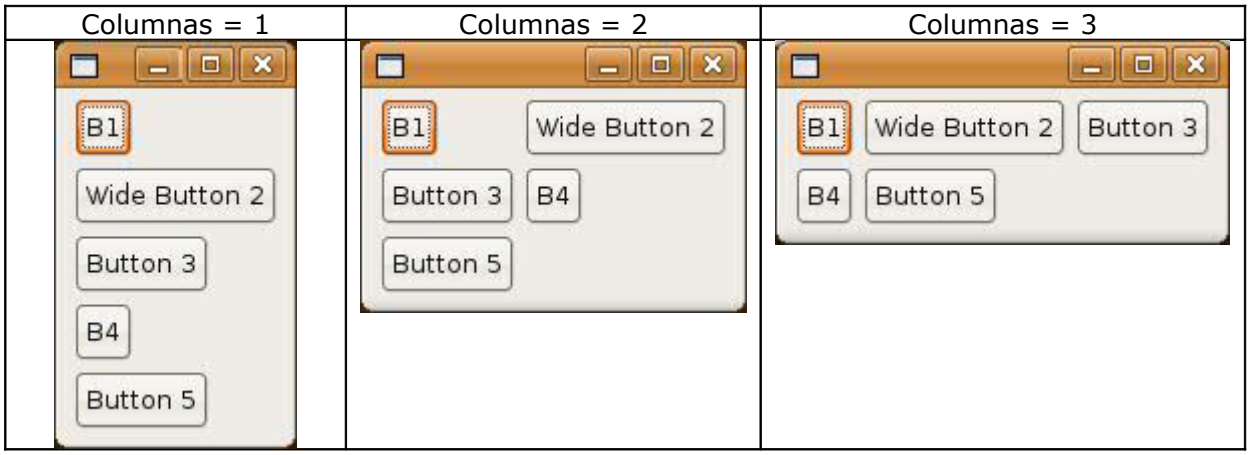

Figura 22. Diagrama funcionamiento GridLayout.

Todas estas herramientas se usaron para crear las ventanas de la aplicación. Primero se creaban las ventanas fuera de ICARO-T y posteriormente se integraban a los agentes.

Estructura de paquetes. Implementación de agentes y recursos.

Para comprender cómo esta estructurado el código y entender su funcionamiento, se van a seguir los pasos necesarios para implementar un nuevo agente con sus recursos.

Paso 1. Se parte de un diseño previo. A partir de los requisitos, de un caso de uso en concreto, y su correspondiente diagrama de secuencia.

Gracias a este diagrama identificamos los agentes y recursos que nos hace falta implementar.

Paso 2. Definiendo el Agente. Usando el diagrama de secuencia y sabiendo que agente/s deben controlar las operaciones, se crea un autómata que refleje el posible comportamiento del agente según el estado en que se encuentra.

Por ejemplo, usando el diagrama de secuencia que se ha visto anteriormente, (Pág. 8) sobre consultar citas de un paciente, el autómata para el agente secretaria sería:

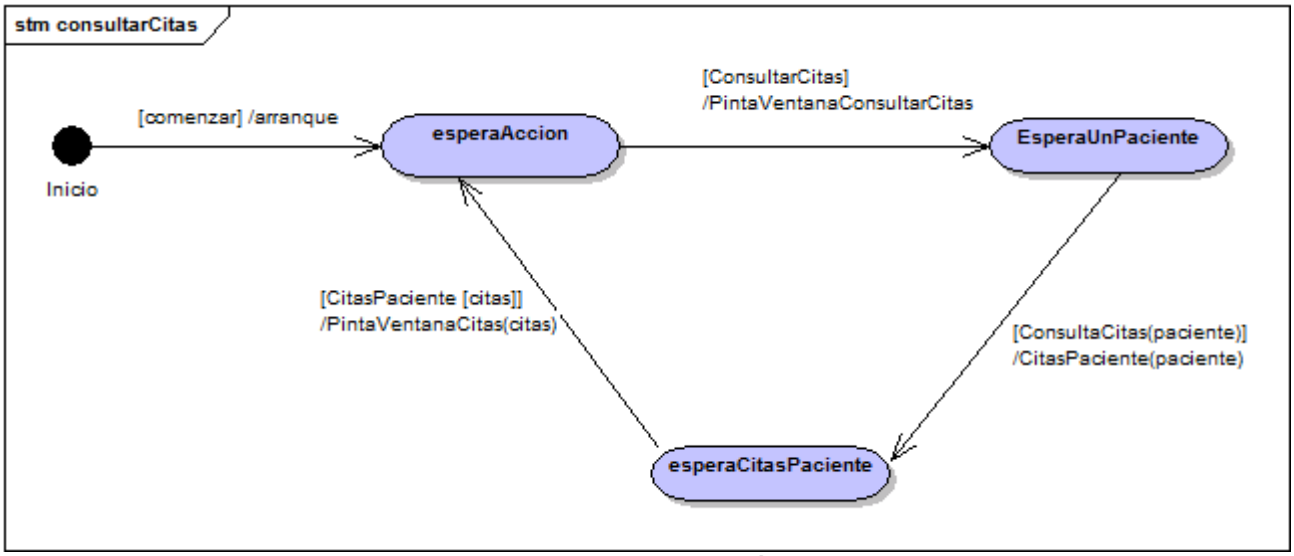

Figura 23. Diseño autómata

Realmente es una versión inicial del autómata ya que luego se le debe añadir funcionalidad y estados adicionales que permitan tratar errores (recuperables o no).

Esta representación gráfica del autómata se traspasa a la implementación a través de la siguiente estructura:

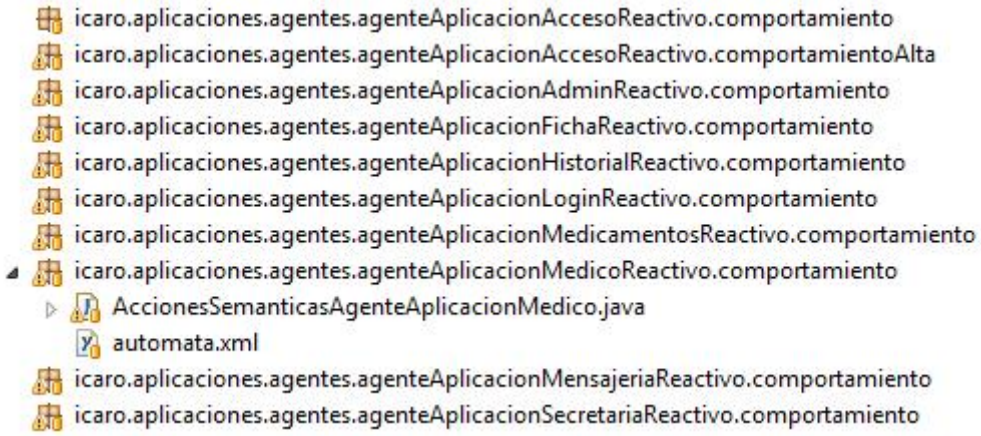

Figura 24. Paquetes del agente.

- **automata.xml**. Es un fichero XML que representa el autómata descrito anteriormente.

```
<tablaEstados descripcionTabla="Tabla para Medico">
    <estadoInicial idInicial="estadoInicial">
        <transicion input="comenzar" accion="nula" estadoSiguiente="empezando" modoDeTransicion="bloqueante"/>
    </estadoInicial>
    <estado idIntermedio="empezando">
        <transicion input="inicio" accion="pintaVentanaMedico" estadoSiguiente="esperaAccion" modoDeTransicion="bloqueante"/>
    </estado>
    <estado idIntermedio="esperaAccion">
        <transicion input="mostrarVentanaMedico" accion="pintaVentanaMedico" estadoSiguiente="esperaAccion" modoDeTransicion="bloqueante"/>
        <transicion input="mostrarTabMed" accion="pintaTabMed" estadoSiguiente="esperaAccion" modoDeTransicion="bloqueante"/>
        <transicion input="mostrarTabMed" accion="pintaTabMed" estadoSiguiente="esperaAccion" modoDeTransicion="bloqueante"/>
        <transicion input="mostrarDatosMed" accion="mostrarDatosMed" estadoSiguiente="esperaAccion" modoDeTransicion="bloqueante"/>
        <transicion input="mostrarMensajes" accion="mostrarMensajes" estadoSiguiente="esperaAccion" modoDeTransicion="bloqueante"/>
        <transicion input="cerrarSesion" accion="nula" estadoSiguiente="empezando" modoDeTransicion="bloqueante"/>
        <transicion input="peticion_terminacion" accion="pedirTerminacionGestorAgentes" estadoSiguiente="esperandoTerminacion" modoDeTransicion="bloqueante"/>
    </estado>
    <estado idIntermedio="esperandoTerminacion">
        <transicion input="termina" accion="terminacion" estadoSiguiente="terminado" modoDeTransicion="bloqueante"/>
    </estado>
    <estado idIntermedio="tratamientoErrores">
        <transicion input="errorRecuperable" accion="nula" estadoSiguiente="esperaFicha" modoDeTransicion="bloqueante"/>
        <transicion input="errorCritico" accion="terminacion" estadoSiguiente="terminado" modoDeTransicion="bloqueante"/>
    </estado>
    <estadoFinal idFinal="terminado"/>
    <transicionUniversal input="termina" accion="terminacion" estadoSiguiente="terminado" modoDeTransicion="bloqueante"/>
    <transicionUniversal input="error" accion="clasifica" estadoSiguiente="tratamientoErrores" modoDeTransicion="bloqueante"/>
</tablaEstados>
```

Figura 25. automata.xml

Esta figura es un extracto del archivo "automata.xml" del agente Médico.

Hay tres tipos de estados: inicial, intermedio y final. Se mencionan una modalidad de transiciones como bloqueante y hay dos tipos de transiciones: normales y universales

- **AccionesSemanticasAgente.** Se trata de la clase que se encarga de implementar su comportamiento. Es decir, cada una de las acciones que puede realizar el agente según refleja su autómata (PintaVentanaConsultarCitas, CitasPaciente(paciente) o PintaVentanaCitas(citas))

AccionesSemanticasAgenteAplicacionMedico.java

```
    /**
     * Muestra la ventana del medico
     */
    public void pintaVentanaMedico(String usuario) {

        try {
            //Se obtiene el visualizador
            visualizacion = (ItfUsoVisualizadorMedico) itfUsoRepositorio.obtenerInterfaz
            (NombresPredefinidos.ITF_USO+"VisualizacionMedico1");

            // Ejemplo de algo que podemos hacer con el
            visualizacion.mostrarVisualizadorMedico(this.nombreAgente, NombresPredefinidos.TIPO_REACTIVO, usuario);

            // Ejemplo de como enviar una traza para asi hacer un seguimiento en la ventana de trazas
            trazas.aceptaNuevaTraza(new InfoTraza(this.nombreAgente,
                    "Se acaba de mostrar el visualizador",InfoTraza.NivelTraza.debug));
        }

        catch (Exception ex) {
            try {
                    trazas.aceptaNuevaTraza(new InfoTraza(this.nombreAgente,
                         "Ha habido un problema al abrir el visualizador de Medico en accion semantica 'pintaVentanaMedico()'",
                        InfoTraza.NivelTraza.error));
                    ex.printStackTrace();
            }catch(Exception e){e.printStackTrace();}
        }
    }
```

Figura 26. Acciones Semánticas del Agente

Esta figura es un extracto real de las accionesSemanticasAgenteMedico. También se puede ver el uso de las trazas. Muy útil a la hora de encontrar errores.

Paso 3. Definiendo Recursos

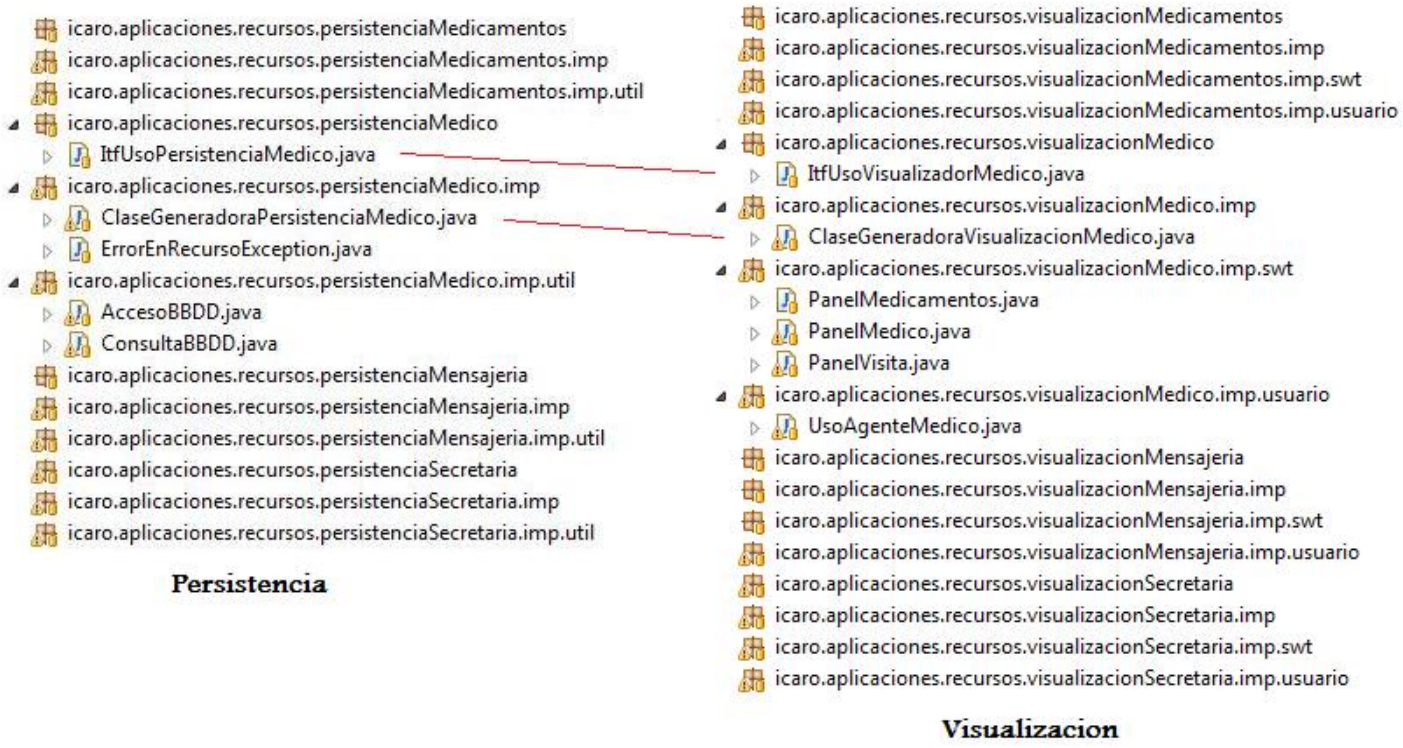

Figura 27. Paquetes para Recursos

El contenido de las carpetas es similar para ambos tipos de recurso:

- Una clase que define la interfaz de uso del recurso
- Visitando el interior de la carpeta "imp" encontramos una clase generadora que se encarga de la creación del modelo computacional del recurso.

Para ambos tipos de recursos siempre se tienen que definir todos los paquetes que aparecen en la imagen superior.

a) Recurso de Visualización

Es el encargado de mostrar las ventanas al usuario.

En el paquete "imp.swt" se mete el código de las ventanas (en este caso usando la librería gráfica SWT) que se han diseñado para comunicarse con el usuario y que competen al agente de este recurso.

En el interior de la carpeta "imp.usuario" se definen las operaciones del recurso que requieren comunicación con el agente y por tanto el envío de eventos, además de posibles pantallas de error.

La clase generadora se encarga de implementar las operaciones que se refieren a las ventanas o paneles que tiene este recurso.

b) Recurso de persistencia

El recurso de persistencia es el encargado almacenar los elementos computacionales de la aplicación. En este caso se ha utilizado como tecnología una base de datos relacional. Por tanto el recurso recubre el acceso a una BD que debe estar instalada en el nodo. Internamente posee una dependencia con un conector SQL.

Las clases "AccesoBBDD" y "ConsultaBBDD" en la carpeta "util", son las encargadas de realizar internamente todas las operaciones que ofrecen las interfaces, ocultando su implementación. La primera realiza la conexión con la BBDD y la segunda implementa las operaciones del recurso sobre la bbdd.

Es la clase generadora la que se encarga de implementar las operaciones relativas al uso de este recurso. Suele usar las operaciones definidas en la clase "ConsultaBBDD".

Paso 4. Clases de dominio.

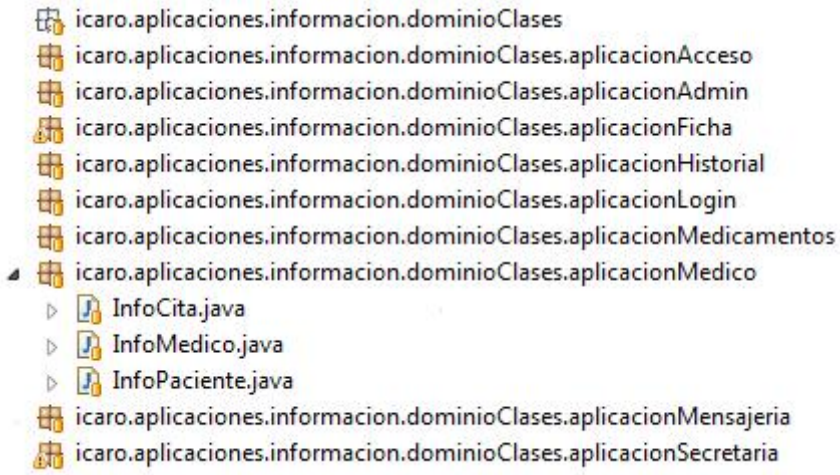

Figura 28. Ejemplo real de las clases de dominio del agente Médico

Son clases que se podrían considerar auxiliares a cada agente y que se definen para el envío de información de eventos. Como se verá en el siguiente punto, los eventos sólo admiten el envío de un dato o una lista de datos del mismo tipo. Estas clases, sirven para unificar en una estructura los datos que se quieren enviar en los eventos a agentes.

Paso 5. Envío de eventos.

Los eventos son entidades contenedoras de información que sirven para desacoplar el proceso de producción de información del consumo por parte de posibles clientes consumidores. Es una entidad de intercambio de información entre el productor del evento y los posibles receptores.

Se utilizan para la comunicación y envío de información de un recurso con su agente o con agentes entre sí.

El formato es el siguiente:

itfUsoNombre.aceptaEvento(new EventoRecAgte(mensaje, datos, origen, Agente_destino));

mensaje: Es una cadena de caracteres que representa un input del autómata del Agente_destino. Este input debe existir y ser exacto al que se especifica aquí. Además, el Agente receptor debe estar en el estado correcto que le permite transitar según el input recibido a otro estado y ejecutar la acción correspondiente.

datos: Puede ser un objeto de un tipo simple, un objeto de los declarados en las clases de dominio o una lista de cualquiera de los dos anteriores. Este campo es opcional. Representa los datos que se van a enviar con el evento.

origen: Identifica al emisor del evento. Puede ser un agente o un recurso.

destino: Identifica al receptor del recurso. Debe ser un agente.

Recurso – agente

UsoAgenteMedico.java

```
/**
 * Activa el agente Historial y abre la lista de visitas
 * @param paciente
 */
public void abrirHistorial(String paciente) {
    getInformacionAgente();

    try {
        if (itfUsoRepositorioInterfaces == null) {
            itfUsoRepositorioInterfaces = ClaseGeneradoraRepositorioInterfaces.instance();
        }

        ItfUsoAgenteReactivo itfUsoHistorial = (ItfUsoAgenteReactivo)itfUsoRepositorioInterfaces.obtenerInterfaz("Itf_Uso_AgenteAplicacionHistorial1");

        itfUsoHistorial.aceptaEvento(new EventoRecAgte("mostrarVentanaLista", paciente, "VisualizacionMedico1", "AgenteAplicacionHistorial1"));

    } catch (Exception e) {
        System.out.println("Ha habido un error al activar el agente Historial desde el agente Medico");
        e.printStackTrace();
    }
}
```

Figura 29. Envío evento recurso –agente

Agente – agente

```
//Una vez comprobado todo correcto se manda a persistencia
agenteSecretaria = (ItfUsoAgenteReactivo) itfUsoRepositorio.obtenerInterfaz
(NombresPredefinidos.ITF_USO+this.nombreAgente);
if(ok){
    persistencia.setCita(datos);
    visualizacion.cerrarVisualizadorCita();
    agenteSecretaria.aceptaEvento(new EventoRecAgte("correcto",this.nombreAgente,NombresPredefinidos.NOMBRE_AGENTE_APLICACION+"Secretaria1"));
}
else{
    ok=false;
    agenteSecretaria.aceptaEvento(new EventoRecAgte("cancelar",this.nombreAgente,NombresPredefinidos.NOMBRE_AGENTE_APLICACION+"Secretaria1"));
    //visualizacion.mostrarMensajeError("Error","Cita incorrecta. Vuelva a intertarlo");
}
```

Figura 30. Envío evento agente –agente

Paso 6. Descripción de la organización.

Agentes, Recursos y Modelo de información constituyen los elementos básicos para construir aplicaciones, pero para que la aplicación funcione utilizando ICARO es necesario definir explícitamente una Organización. La organización que implementa el sistema se define con un documento formal –expresado en XML- donde se especifican las propiedades y los componentes de la organización, los roles, atributos, características y dependencias de los componentes. Este documento se utiliza para crear la organización, es decir para crear los componentes computacionales que deben llevar a cabo la funcionalidad del sistema.

```
<!--*****************************************************************************************************
*****************     Propiedades globales de la organización                          *****************
*****************************************************************************************************-->
<icaro:PropiedadesGlobales>
<icaro:intervaloMonitorizacionGestores>5000</icaro:intervaloMonitorizacionGestores>
<icaro:activarPanelTrazasDebug>true</icaro:activarPanelTrazasDebug>
    <icaro:listaPropiedades>
        <icaro:propiedad atributo="gestores.comun.intervaloMonitorizacion"
            valor="50000" />
    </icaro:listaPropiedades>
</icaro:PropiedadesGlobales>
<icaro:DescripcionComponentes>
    <icaro:DescComportamientoAgentes>
<!--*****************************************************************************************************
*****************  Descripción del comportamiento de los gestores                   *****************
*****************************************************************************************************-->
        <icaro:DescComportamientoGestores>
            <icaro:DescComportamientoAgente
                nombreComportamiento="GestorOrganizacion" rol="Gestor" tipo="Reactivo" />
            <icaro:DescComportamientoAgente
                nombreComportamiento="GestorAgentes" rol="Gestor" tipo="Reactivo" />
            <icaro:DescComportamientoAgente
                nombreComportamiento="GestorRecursos" rol="Gestor" tipo="Reactivo" />
        </icaro:DescComportamientoGestores>
<!--*****************************************************************************************************
*****************   Descripción del comportamiento de los agentes de aplicación     *****************
*****************************************************************************************************-->
        <icaro:DescComportamientoAgentesAplicacion>
            <icaro:DescComportamientoAgente
                nombreComportamiento="AgenteAplicacionLogin" rol="AgenteAplicacion"
                tipo="Reactivo" />
            <icaro:DescComportamientoAgente
                nombreComportamiento="AgenteAplicacionMedico" rol="AgenteAplicacion"
                tipo="Reactivo" />
            <icaro:DescComportamientoAgente
                nombreComportamiento="AgenteAplicacionHistorial" rol="AgenteAplicacion"
                tipo="Reactivo" />
            <icaro:DescComportamientoAgente
                nombreComportamiento="AgenteAplicacionMedicamentos" rol="AgenteAplicacion"
                tipo="Reactivo" />
            <icaro:DescComportamientoAgente
                nombreComportamiento="AgenteAplicacionSecretaria" rol="AgenteAplicacion"
                tipo="Reactivo" />
            <icaro:DescComportamientoAgente
                nombreComportamiento="AgenteAplicacionFicha" rol="AgenteAplicacion"
                tipo="Reactivo" />
            <icaro:DescComportamientoAgente
                nombreComportamiento="AgenteAplicacionAdmin" rol="AgenteAplicacion"
                tipo="Reactivo" />
        </icaro:DescComportamientoAgentesAplicacion>
    </icaro:DescComportamientoAgentes>
<!--*****************************************************************************************************
*****************   Descripción de los recursos de aplicación                       *****************
*****************************************************************************************************-->
    <icaro:DescRecursosAplicacion>
        <icaro:DescRecursoAplicacion nombre="VisualizacionLogin"/>
        <icaro:DescRecursoAplicacion nombre="PersistenciaLogin"/>
        <icaro:DescRecursoAplicacion nombre="PersistenciaMedico"/>
        <icaro:DescRecursoAplicacion nombre="VisualizacionMedico"/>
        <icaro:DescRecursoAplicacion nombre="VisualizacionHistorial"/>
        <icaro:DescRecursoAplicacion nombre="PersistenciaHistorial"/>
        <icaro:DescRecursoAplicacion nombre="VisualizacionMedicamentos"/>
        <icaro:DescRecursoAplicacion nombre="PersistenciaMedicamentos"/>
        <icaro:DescRecursoAplicacion nombre="PersistenciaSecretaria"/>
        <icaro:DescRecursoAplicacion nombre="VisualizacionSecretaria"/>
        <icaro:DescRecursoAplicacion nombre="PersistenciaFicha"/>
        <icaro:DescRecursoAplicacion nombre="VisualizacionFicha"/>
        <icaro:DescRecursoAplicacion nombre="VisualizacionAdmin"/>
        <icaro:DescRecursoAplicacion nombre="PersistenciaAdmin"/>
    </icaro:DescRecursosAplicacion>
</icaro:DescripcionComponentes>
<icaro:instancias>
```

Figura 31.Parte de la descripción de la organización del sistema.

La organización tiene una serie de propiedades computacionales globales aplicables al conjunto de sus componentes. Las que se especifican en el documento se refieren al intervalo de monitorización de los gestores, a la activación o no de las trazas para comprobar el comportamiento de los componentes. Pueden añadirse nuevas propiedades siguiendo el formato atributo valor.

A continuación se describen los Componentes de la organización. Se distinguen tres tipos de componentes: Gestores de la Organización. Agentes de aplicación y recursos de aplicación.

La organización tiene tres gestores: El Gestor de la Organización, el Gestor de Agentes y el Gestor de Recursos. Los gestores son agentes cuyo rol en la organización esta predefinido, es decir deben dedicarse a gestionar otras entidades.

Para cada agente es necesario definir su comportamiento, es decir la información que va a utilizarse para crear su modelo computacional. Los agentes de aplicación se describen de forma similar a los agentes gestores.

Los recursos se describen especificando el identificador del recurso y la localización de la clase que permite la creación del recurso. Si no se especifica la localización se buscará en la ruta por defecto.

Una vez definidos los componentes se especifican los ejemplares de agentes y recursos que implementarán la funcionalidad de la aplicación. Ya que puede haber varias instancias de cada agente o recurso.

Base de datos

La implementación de la base de datos se ha realizado usando MySQL. Se eligió MySQL como tecnología para gestionar la base de datos por su buen rendimiento y facilidad de uso, además de por ser un sistema libre y gratuito.

La comunicación de la aplicación con MySQL se ha hecho usando una librería de Java llamada JDBC.

Para comunicar los agentes con la base de datos ha sido necesario crear recursos de persistencia. Cuando un agente necesita leer o escribir en la base de datos le envía un evento a un recurso de persistencia que se ocupa de la comunicación. De esta manera se puede cambiar fácilmente la base de datos sin que el agente se entere. Por ejemplo, se podría cambiar de MySQL a Oracle y lo único que habría que hacer es actualizar el recurso de persistencia. Esto es algo totalmente transparente a los agentes.

Se barajaron dos posibles formas de comunicarse con la base de datos: Centralizada o distribuida.

En un principio se optó por tener el acceso a la BD de forma centralizada, con un agente propio que se ocupara de gestionar la conexión. Parecía mejor opción ya que así se evitaba tener múltiples conexiones a la base de datos, pero finalmente se optó por una comunicación distribuida. A continuación se pueden ver la lista de pros y contras que se tuvieron en cuenta al tomar la decisión:

Centralizada
Pros:
- Una sola conexión. El tiempo de conexión suele ser lo que mas tarda en un acceso a la base de datos.
- Todos los métodos de acceso juntos. Cualquier agente puede hacer cualquier consulta.

Contras:
- Requiere la creación de un agente gestor de base de datos.
- Para cada consulta a la base de datos hay que enviar un evento al agente gestor.
- Interfaz de uso con muchos métodos.
- Todos los agentes dependen del gestor.

Distribuida
Pros:
- Interfaz de uso con solo los métodos específicos de cada agente (pocos métodos por recurso de persistencia).
- Para cada consulta tan solo hay que usar el agente propio.
- La creación de cada recurso de persistencia se podría hacer con un Template.
- Hace más independiente a cada agente.

Contras:
- Hay que crear un recurso de persistencia en cada agente.
- Muchas conexiones a la base de datos. Esto solo supone un problema de tiempo de acceso, pero solo al cargar la aplicación. Una vez establecidas todas las conexiones, MySQL puede soportar un número elevado de conexiones concurrentes sin problemas.

Figura 32. Comunicación centralizada o distribuida con la base de datos.

Se decidió usar la comunicación distribuida sobretodo por el hecho de que los agentes fueran los más independientes posible y que los recursos de persistencia fueran ligeros.

También se tuvo en cuenta que la comunicación con la persistencia se hace de forma síncrona, mientras que la comunicación entre agentes suele ser asíncrona. Esto significa que cuando se accede a un recurso de persistencia, este se bloquea hasta que devuelve un resultado.

Si cada agente tiene su recurso de persistencia, por mucho que este se bloquee no afecta a los demás agentes cuando quieran acceder a la base de datos.

Seguridad y Confidencialidad

En una aplicación como esta es necesario garantizar la seguridad y confidencialidad de la información de los pacientes. Durante la implementación se ha tenido en cuenta esto y se han llevado a cabo varias medidas:

1. Sistema de usuarios: Hay 4 tipos de usuarios. Cada usuario solo puede acceder a una determinada parte de la base de datos.
   - Medico: Los usuarios de tipo 'Médico' son los únicos que pueden ver los datos médicos de un paciente.
   - Secretaria: Las secretarias tan solo pueden acceder a la agenda y a los datos personales de un paciente.
   - Paciente: Puede acceder a sus propios datos pero no a los de otros pacientes ni a la información de la agenda.

- Administrador: No puede acceder a ningún tipo de información sobre los pacientes. Tan solo puede realizar tareas de configuración y administración de la base de datos.

2. Cada tipo de usuario está controlado por un agente específico. Por ejemplo, el agente 'Secretaria' tiene implementada tan solo su propia funcionalidad. No hay manera de que desde la interfaz de usuario se pueda acceder a la funcionalidad del agente Médico. En otras palabras, cada agente tiene su funcionalidad aislada lo cual mejora la seguridad.

3. Los recursos de persistencia son locales para cada agente y tienen implementados solo los métodos de acceso a los datos específicos de cada usuario.

Pruebas.

El diseño de pruebas ha estado presente desde el inicio de la implementación. Con cada nueva funcionalidad añadida a la aplicación, se realiza una prueba local de dicho módulo y posteriormente una prueba global con el conjunto de la aplicación.

Este sistema de pruebas además es verificado en primer lugar por el programador que lo implementó y posteriormente por otro, ajeno al diseño de dicho módulo. En este caso, con el compañero de proyecto.

Para automatizar un poco todo este sistema se ha diseñado una plantilla que detalla el resultado obtenido en cada prueba, de forma que cualquier incidencia queda claramente reflejada.

Test-Id	
Descripción	
Entrada	
Actores	
Objetivo	
Fallos posibles	
Pasos para la prueba. Acciones	
Id	**Descripción**
1	
2	
3	
4	
5	
Ejecución de la prueba	
Resultado	
Comentarios	
Fecha	
Autor de la prueba	

Tabla 2. Plantilla pruebas.

Semanalmente se ha dado un repaso a las pruebas para corregir errores y verificar la integridad del sistema.

Para más detalles de las pruebas se puede consultar el anexo III, Pág. 63

Experimentación

Comunicación externa con ICARO

Aparte de la aplicación instalable, también se quiere dar la opción de acceder a la aplicación de forma remota, ya sea desde una página web o desde un móvil.

Aquí es donde se aprovechan las ventajas que da un sistema multiagente. ICARO-T fuerza a hacer una separación entre la lógica y las interfaces. Esto al final facilita que la lógica pueda ser reutilizada desde distintas interfaces. Se trata de aprovechar los agentes ya programados, desde fuera de la aplicación en si. De esta manera se puede usar la aplicación en otros sistemas sin tener que volver a programar la misma funcionalidad en otro lenguaje. Además, se mantiene el control que tiene ICARO-T sobre los agentes y seguirían funcionando tanto la maquina de estados como el sistema de trazas.

Por ejemplo, si se quiere hacer una página web para que la secretaria pueda acceder a su agenda, no se tendría que volver a programar en un lenguaje web los accesos de la secretaria a la base de datos para consultar la agenda.

Bastaría con que la página web se comunique mediante Servlets con una aplicación Java residente en el servidor. Esta aplicación Java se ocupa de comunicarse con la instancia de la aplicación que está ejecutándose, para pedirle la agenda al agente secretaria. Finalmente el agente secretaria devolvería el resultado a la aplicación intermedia para que esta se lo envíe a la interfaz web.

En caso de no haber usado agentes, se tendría que volver a programar la funcionalidad de secretaria en un leguaje web (por ejemplo, PHP) para permitir al usuario hacer lo mismo que con la aplicación. Esto no solo supone hacer lo mismo dos veces, sino que hace que se pierda el control que se esta haciendo en cada momento y aumenta las posibilidades de error.

Separación de Agentes y Recursos

Una de las posibilidades que ofrece un sistema multiagente es la modularidad. Cada agente debe ser independiente de los demás. Esto significa que una aplicación compuesta por varios agentes podría convertirse en una aplicación distribuida entre varias máquinas. Puede haber varias razones para hacer esto, por ejemplo para optimizar el rendimiento de la aplicación.

Se experimentó con esta idea para ver si se podía separar los agentes manteniendo la funcionalidad de ICARO-T. La idea era la siguiente: En cada instancia de la aplicación hay una instancia de ICARO-T ejecutándose con unos determinados agentes. Estos agentes se han probado juntos en una misma máquina y funcionan. Sin cambiar nada deberían funcionar también separados.

Por ejemplo, en una máquina está el agente *médico* y en otra el agente *mensajería*. Los agentes deberían funcionar exactamente igual que si estuvieran juntos, no tienen por qué cambiar su funcionamiento interno. Es más, ni siquiera se darían cuenta de que le están enviando un evento a un agente que está en otra máquina.

Para conseguir esta funcionalidad hace falta algún tipo de software que permita la comunicación entre componentes. Entre los que se ha estudiado están CORBA, RMI y Servlets. CORBA y RMI son muy similares en su funcionamiento, mientras que los Servlets son ligeramente distintos ya que solo se usan en un contexto Web.

El fundamento sobre el que se basan CORBA y RMI es el de la invocación de objetos remotos. Para ello hace falta definir una estructura cliente servidor.

- En el lado del servidor hay que crear los objetos remotos. Para crear un objeto remoto, se define una interfaz y el objeto remoto será una clase que implemente dicha interfaz.
- En el lado del cliente simplemente hay que buscar el objeto remoto en el registro de la máquina remota. Tras esto se puede usar igual que cualquier método definido en la interfaz.

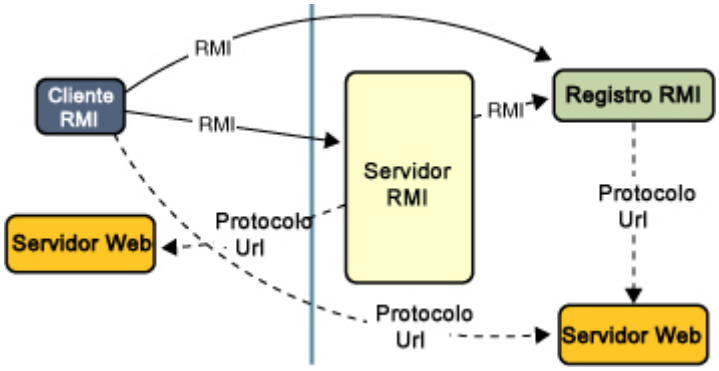

Figura 33. Funcionamiento RMI

La diferencia entre CORBA y RMI es que la primera permite la comunicación entre objetos de distintos lenguajes de programación, mientras que RMI solo permite la comunicación entre objetos Java. Finalmente se decidió investigar sobre el funcionamiento de RMI ya que la aplicación que se está desarrollando utiliza solamente Java.

El principal problema que se tuvo que afrontar es que el uso de RMI implica adaptar los componentes para que puedan comunicarse. Esto en una aplicación pequeña no suele suponer una gran dificultad ya que basta con añadir una interfaz sencilla e implementarla, pero no es así en este caso. Hay que añadir una interfaz manteniendo el buen funcionamiento de ICARO-T que ya de por sí tiene sus interfaces de uso y de gestión.

Versión portable

A la hora de decidir como será la distribución de la aplicación se han barajado 3 opciones:

- Versión instalable
- Versión portable
- Versión remota

La versión remota se ha descartado ya que no se han conseguido los resultados esperados con RMI. Esta opción era interesante porque permitía que la aplicación fuera fácilmente actualizable sin necesidad de que el usuario final cambie nada en su ordenador.

Una versión instalable es una opción más clásica. Distribuir un instalador que se ocupe de copiar los archivos al disco duro del usuario, configurar el entorno y crear accesos directos. Esta opción no es mala pero tampoco es necesario obligar al usuario a instalar la aplicación cada vez que quiera acceder a la aplicación desde una máquina distinta.

Otra opción es hacer una versión portable que se pueda llevar a cualquier sitio fácilmente y se pueda ejecutar en cualquier máquina sin necesidad de instalación. Esta opción es muy interesante y Java da muchas facilidades para ello. Basta con distribuir el ejecutable .jar y una versión de Java compatible. En realidad, debería ser suficiente con el .jar pero se ha visto que una misma aplicación Java puede dejar de funcionar si no hay una versión de Java compatible.

La única limitación que podría surgir para una versión portable es que la aplicación ocupe mucho espacio, por lo cual ya no sería tan portable. Calculando el espacio que ocupa la aplicación + el espacio que ocupa Java, se ha visto que en menos de 60MB cabe todo. Viendo la capacidad de las memorias usb existentes en la actualidad, se ha llegado a la conclusión de que 60MB es más que razonable para una aplicación portable.

Si además le sumamos que la aplicación usa una base de datos remota, no queda duda de que es una gran ventaja esta última opción. Con una versión portable el médico ya no tiene que preocuparse de tener una estación de trabajo. Puede ejecutar la aplicación desde la consulta, desde su casa o incluso desde un cyber-cafe.

Resultados y conclusiones

Métricas.

Durante el curso se ha ido tomando nota de las horas que se han invertido en el proyecto, divididas por categorías. Esto se ha hecho con el fin de poder hacer un análisis al final para poder sacar conclusiones. Estas conclusiones servirán como experiencia para en un futuro saber cuales han sido los aspectos del proyecto que más tiempo lleva realizar, donde hay que enfocar el esfuerzo y como debe planificarse un proyecto. En otras palabras, se trata de aprender de la experiencia ganada con el proyecto para aplicarla en trabajos futuros.

En el gráfico siguiente se puede ver la distribución de las horas invertidas, separadas en distintas categorías:

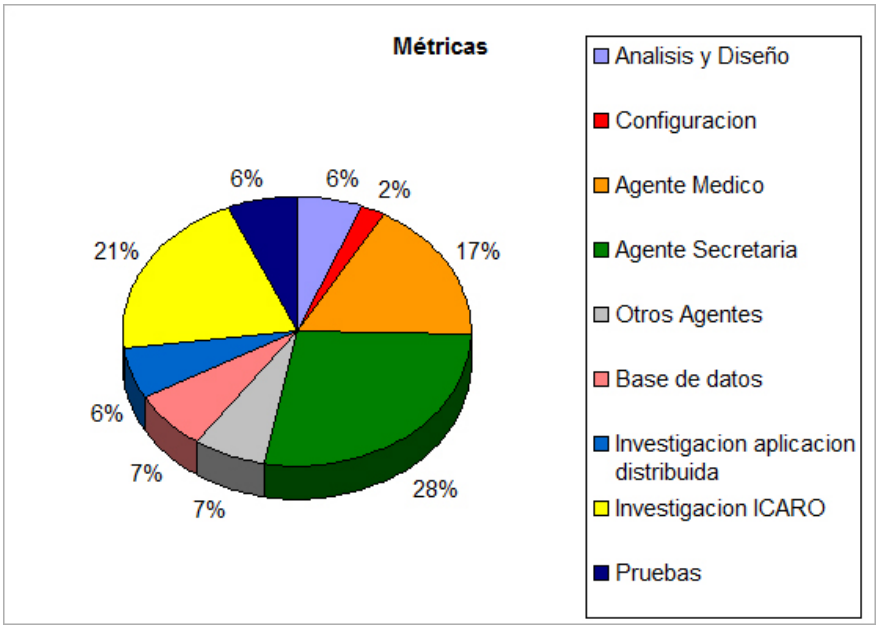

Figura 34. Gráfico métricas.

De este gráfico se pueden sacar varias conclusiones:

- Un 27% del tiempo se invierte en investigación. Gran parte de ella se ha invertido en investigación del funcionamiento de la infraestructura que se va a utilizar, es decir, en la investigación inicial de la base sobre la que se va a desarrollar la aplicación. Esto es algo a tener muy en cuenta ya que un tercio del tiempo total del proyecto se invierte en investigación.

  En un inicio no se pensó que esta parte del trabajo fuera tan costosa. Se puede decir que la curva de aprendizaje es una parte muy importante de un proyecto. Este periodo tan largo de investigación y aprendizaje se llevó a cabo pensando en que más adelante se pueda amortizar aprovechando las ventajas que proporciona un sistema multiagente. Sobretodo por la reutilizabilidad que ofrecen los agentes y la portabilidad de ICARO-T al estar realizado en un lenguaje multiplataforma (Java).

- El análisis de los requisitos y el diseño inicial de la aplicación se lleva un 6% del tiempo. Esto puede parecer poco pero no lo es ya que son bastantes horas. En

este caso son 20 horas y prácticamente todo este tiempo se usa durante el primer mes de trabajo.

- Como es lógico, la mayor parte del tiempo la ocupa el desarrollo de la aplicación. En este caso es un 59% del tiempo que se invierte en total entre agentes y base de datos. La gran mayoría en lo que es programar la aplicación en sí.

  La base de datos tan solo ocupa un 7% del tiempo ya que la mayoría de trabajo se hace en su fase de diseño. Una vez diseñada la base de datos la implementación es muy sencilla ya que tras crear la estructura relacional en código SQL no hay mucho más que hacer.

- La fase de pruebas es un 6% del tiempo total pero no es algo que se haga de forma constante durante todo el proyecto. Gran parte del tiempo en pruebas se invierte en los últimos meses de desarrollo.

- La configuración del entorno no debe llevar mucho tiempo. Si se usa un entorno ya conocido se ahorran horas de trabajo y se puede empezar a trabajar cuanto antes. En nuestro caso es solo un 2% del total pero se ha perdido aquí más tiempo del esperado.

- Resulta interesante observar el desarrollo de dos agentes ha requerido un 45% del tiempo mientras que los demás agentes (son cinco más) tan solo llevan el 7%. Esto muestra que el esqueleto de un proyecto requiere muchas horas. En nuestro caso el esqueleto son el agente médico y el agente secretaria. A partir de estos dos agentes se llaman a todos los demás.

Problemas encontrados y solución adoptada

1. Configuración de ICARO con Eclipse

En la configuración del entorno se invirtieron entre 6 y 8 horas. Es un tema que en un principio no debía suponer problemas pero no fue así. Se tomó la decisión de usar Eclipse como entorno de desarrollo por ser un entorno con el que ya se tenía mucha experiencia.

El primer problema que se encontró fue que ICARO-T está configurado para funcionar correctamente con Netbeans. No había sido probado con Eclipse. Durante el proceso de migración de la infraestructura, se encontraron varios fallos generalmente causados por fallos en la configuración del proyecto (configuración ausente ya que la configuración de Netbeans no se puede importar en Eclipse). Había que importar las librerías que usa ICARO-T, configurar las rutas y finalmente configurar la ejecución.

Una vez que se consiguió un correcto funcionamiento de ICARO-T en Eclipse todo fue mucho más fácil. Tan solo se volvieron a encontrar problemas cuando salió una nueva versión de ICARO-T.

2. Migración de ICARO-T a IcaroMini

En el mes de Abril salió una nueva versión de ICARO-T llamada IcaroMini. Siguiendo el consejo del tutor se decidió hacer una migración de la aplicación a IcaroMini. No es tan trivial como parece ya que en la nueva versión se cambió el nombre a algunas librerías y algunos métodos. El primer intento de migración fue fallido debido a un error que daba uno de los paquetes de IcaroMini en Eclipse.

Un mes después se volvió a intentar la migración con una nueva versión de IcaroMini. Esta vez todo funcionó bien y tan solo hubo que invertir unas dos horas renombrando las llamadas a las librerías y a los métodos cuyos nombres habían cambiado. Tras esto se

probó la aplicación y se puede decir que funciona correctamente sin tener que hacer ningún cambio más al código ya programado.

3. Sistemas multiagente e implementación con ICARO-T

Se podría decir que la investigación de cómo funciona un sistema multiagente y más concretamente de como funciona ICARO-T es lo que más tiempo ha requerido.

Se tardó aproximadamente dos semanas en conseguir entender el funcionamiento de un sistema multiagente, cómo se implementa y cuales son sus ventajas. Una vez que se comprendió esto, se elaboró una lista inicial de agentes y recursos. A continuación se empezó a hacer pruebas con ICARO-T para ver como funciona la infraestructura que se debía usar.

Desde el momento en que se empezaron las pruebas hasta que se empezó la aplicación pasaron unos dos meses y se invirtieron unas 40 horas. Esto se debe a varias razones.

- No hay un punto de partida:

  En un principio no se sabe por donde empezar. La documentación de ICARO-T da una explicación muy detallada de su funcionamiento, con un ejemplo práctico y con diagramas de componentes, pero la explicación es tan detallada y tan técnica que acaba confundiendo. Es necesario un punto de partida. Algo más corto pero más directo, algo con lo que se pueda entender de forma global los componentes de cada agente y recurso antes de entrar a mirar los detalles.

- Falta de conocimiento:

  El hecho de no conocer lo que era un sistema multiagente hace que sea más difícil entender el funcionamiento de los agentes en ICARO-T. Se tienen que hacer muchas pruebas hasta que se llega a entender como se comunica un agente con un recurso o un agente con otro agente. Muchas veces cosas que ahora se ven muy sencillas al principio se veían muy complicadas.

- Sensibilidad del sistema:

  ICARO-T está hecho de tal manera que obliga al desarrollador a programar de forma estrictamente correcta. Cualquier pequeño error o warning se magnifica con ICARO-T provocando en muchas ocasiones excepciones o bloqueos. Esto hace que durante la fase de aprendizaje se pierda mucho tiempo intentando averiguar porque no funciona un cierto código que sí funciona fuera de ICARO-T o porque se bloquea el sistema al cambiar unas líneas que aparentemente no suponen un gran cambio en el agente.

- Incompatibilidad con otras librerías:

  ICARO-T es un sistema bastante complejo en el que hay muchos threads ejecutándose a la vez, cada uno realizando su tarea, pero todos controlados y monitorizados por un gestor de agentes. Durante la ejecución de una aplicación se están mandando mensajes y objetos constantemente de un thread a otro. El problema es que esos mensajes no son directos. Para llegar del origen al destino hay que pasar por muchas fases intermedias. Todo esto hace que ciertas librerías no funcionen de forma correcta.

  En concreto hay una librería muy importante que ha estado dando muchos problemas durante todo el desarrollo y que ha hecho perder mucho tiempo al principio hasta encontrar la manera de hacerla funcionar con ICARO-T. Se trata de la librería SWT,

que se usa para la interfaz grafica. Es una librería que se quería usar ya que da buenos resultados pero que daba excepciones constantemente. Con ICARO-T se ha probado Swing con buenos resultados pero SWT nunca había sido probada. Finalmente se tuvo que encapsular al máximo cada uno de los threads de SWT para conseguir que funcionara correctamente pero aun así hay cierta funcionalidad que no se ha podido usar.

- Archivos XML de configuración:

En ICARO-T el archivo de configuración es muy importante y muy sensible a fallos. Ahí es donde se declaran los agentes, los recursos, las instancias y la configuración de la conexión a la base de datos. Es muy fácil cometer un error en este archivo ya que hay que poner los nombres exactamente igual a como se llaman en el código. En caso de cometer algún fallo la aplicación falla y no es fácil encontrar el error. Es difícil darse cuenta de que el error está en el archivo de configuración, sobretodo al principio.

Tras un periodo difícil de aprendizaje se llegó a comprender el funcionamiento de ICARO-T pero se vio que el proceso para crear un nuevo agente era muy repetitivo y costoso en tiempo. Por ello se tomó la decisión de crear un "Template" (una plantilla). Esta plantilla proporcionó muchísimas ventajas y facilitó la tarea. A continuación se listan algunas de las ventajas:

- La creación de un nuevo agente se puede hacer en cuestión de minutos con el template. Basta con cambiar el nombre a los paquetes y las clases. Una vez hecho esto se tiene un agente funcionando correctamente con su autómata, su clase generadora, su recursos de visualización y sus eventos ya creados.

- Hay una serie de errores comunes que por despistes o por desconocimiento se cometen muchas veces. Con el template se evitan estos errores ya que se está usando un código probado y optimizado. Según se va desarrollando la aplicación, el template se va optimizando más, por lo que con el paso del tiempo se tiene una base muy sólida sobre la que empezar.

- En el template se incluyen unos ejemplos muy básicos de como funciona la comunicación entre agentes y recursos. Esto hace que sea mas fácil entender el funcionamiento y evita que se invierta tiempo intentando descifrar como funciona una aplicación mas grande ya hecha.

4. Integración de ICARO-T con tecnologías web o acceso remoto

Una vez que la aplicación se encontraba en un estado estable, se empezó a investigar sobre cómo usar la aplicación vía web y cómo separar los agentes para un uso distribuido de la aplicación.

En un principio se optó por usar una herramienta llamada GWT para acceder mediante RPC (llamadas remotas a procedimientos) a la aplicación. La idea era lanzar una instancia de ICARO-T cada vez que un usuario accedía vía web. En este proceso se invirtieron unas 8 horas sin resultados positivos ya que no se podía arrancar ICARO-T desde un servidor web.

Lo siguiente fue usar Servlets con una aplicación Java intermedia que se comunicara con ICARO. Se dedicaron unas 10 horas en la investigación de lo que es un servlet, cómo funciona y finalmente cómo usarlo para acceder a ICARO remotamente. Finalmente no se consiguió que funcionara correctamente ya que hacia falta una aplicación intermedia que se ejecutara en el servidor web para recibir las llamadas desde la página web y redirigirlas a una instancia de ICARO-T.

Finalmente se pensó en usar RMI, una tecnología Java que permite hacer llamadas a objetos remotos. Esta solución es la que parece que puede dar mejores resultados pero con una complejidad añadida: Hay que adaptar los componentes creando nuevos interfaces. En la investigación del funcionamiento de RMI se invirtieron unas 10 horas más. No se ha podido hacer pruebas funcionales por falta de tiempo pero se podría concluir que este es el camino que hay que seguir para conseguir resultados positivos.

5. Acceso a la aplicación vía móvil

El acceso desde móvil es algo que puede ser muy interesante en un futuro. Se invirtieron de 8 a 10 horas investigando cual es la mejor manera de implementar esta funcionalidad de manera que sea compatible con la aplicación desarrollada.

Hay dos opciones. Una es hacer una aplicación en el móvil usando J2SE que se comunique vía Internet con la aplicación en ICARO-T. Viendo las dificultades que supone el acceso remoto en desde un ordenador normal se descarto esta opción.

La otra opción es usar las facilidades que da un teléfono móvil para acceder a la web para así desde una página web adaptada al móvil acceder a la aplicación de forma similar a la descrita más arriba.

Versiones futuras

La aplicación se seguirá desarrollando en el futuro y hay cierta funcionalidad que se podría añadir usando como base el trabajo y la investigación ya realizados.

Estas son algunas de las ideas que se tienen:

- Aplicación Web: Dar acceso a la aplicación desde un navegador web. No solo a médico y secretaria sino también a los pacientes. Se quiere permitir al paciente consultar sus citas, enviar mensajes al médico o consultar el estado de la consulta para así saber el tiempo de espera aproximado que hay en un momento determinado.

- Aplicación móvil: Permitir el acceso a la aplicación desde el navegador de un teléfono móvil. La funcionalidad sería como la de la aplicación web pero quizás un poco más reducida ya que hay que ajustarse a las capacidades de un teléfono móvil.

  Sería muy interesante que un paciente pueda consultar en cualquier momento desde su móvil el tiempo de espera que hay en la consulta para así poder aprovechar mejor su tiempo. Incluso se podría plantear el que un paciente pueda hacer una foto con el móvil, de una herida (por ejemplo) y enviársela directamente al médico.

- Aplicación distribuida: Aprovechando la investigación hecha sobre RMI se podría hacer pruebas con la aplicación distribuida para ver si mejora el rendimiento. Poniendo algunos agentes en maquinas optimizadas solo para una tarea podría mejorar el rendimiento de la aplicación.

- Compatibilidad de idiomas: Se tiene pensado que la aplicación pueda ser traducida fácilmente a cualquier idioma. Para ello hay que idear un sistema por el cual el texto no esté metido directamente en el código de cada agente sino que se haga una consulta externa. De esta manera se puede traducir la aplicación sin modificar el código. Incluso podría traducirla alguien que no tenga ni idea de programación.

Bibliografía

- **Transparencias SMA.ppt,** Juan Pavón Mestras
- **ManualUsiIcaroMini V1.1** Francisco J Garijo, Felipe Polo, Damiano Spina
- **Manual de servlet.** http://www.javapassion.com/j2ee/ServletBasics/ speakernoted.pdf
- **Consultas librería SWT**. http://www.java2s.com
- **Wikipedia** en general, para distintos conceptos. http://www.wikipedia.org
- **Documentación Java**, http://www.java.org
- **Documentación Eclipse**, http://www.eclipse.org
- **Documentación MySQL**, http://www.mysql.com
- **Tutorial RMI** de Sun, http://java.sun.com/docs/books/tutorial/rmi/index.html
- **Apuntes de la asignatura BDSI** para el diseño de la BD, Rafael Caballero Roldán
- **Utilización de UML en ingeniería del software con objetos y componentes**, Perdita Stevens, Rob Pooley

Glosario

SMA: Sistema multiagente. Es un sistema distribuido en el cual los nodos o elementos son sistemas de inteligencia artificial, o bien un sistema distribuido donde la conducta combinada de dichos elementos produce un resultado en conjunto *inteligente*.

Agente: Demazeau, define el concepto de agente desde el punto de vista externo e interno. Desde el punto de vista externo, "Agente es una entidad real o virtual que evoluciona en un entorno, es capaz de percibir y sobre ello actuar sobre este entorno, capaz de comunicarse con otros agentes y que exhibe un comportamiento autónomo", mientras que desde el punto de vista interno es "una entidad real o virtual con control local en algunos de sus procesos de percepción, comunicación, adquisición de conocimiento, razonamiento, decisión, ejecución y acción.

Sistema Distribuido: Un sistema separado en componentes que trabajan juntos pero se ejecutan en distintas máquinas.

Ficha: Datos personales de un paciente.

Cita: Fecha y hora asignada a un paciente para ser visto por un médico.

Historial: Conjunto de visitas de un determinado paciente

Visita: Resultado de un paciente tras ser atendido por un médico.

Receta: Conjunto de medicamentos que un médico prescribe a un paciente.

Índice de Tablas y Figuras

Tabla 1. Riesgos 8

Tabla 2. Plantilla pruebas..... 42

Figura 1. Diagrama Secuencia ConsultarCitasPaciente. 9

Figura 2. Diagrama Secuencia CrearFicha. 10

Figura 3. Diagrama de secuencia para sacar interfaces. 11

Figura 4. Ejemplo de interfaz 1. 13

Figura 5. Ejemplo de interfaz 2. 14

Figura 6. Ejemplo de interfaz 3. 15

Figura 7. Diagrama Entidad-Relación de la base de datos..... 16

Figura 8. Concepto SMA 17

Figura 9. Concepto SMA. Monitorización Agente. 18

Figura 10. Concepto SMA. Acciones Agente 18

Figura 11. Concepto SMA. Ejecución tareas, comunicación entre agentes 19

Figura 12. Concepto SMA. Agentes y sus recursos 20

Figura 13. Arquitectura Sistema. Diagrama arranque de la aplicación 23

Figura 14. Arquitectura Sistema. Agente administrador 24

Figura 15. Arquitectura Sistema. Agente Secretaria. 25

Figura 16. Arquitectura Sistema. Agente Médico. 26

Figura 17. Arquitectura ICARO 27

Figura 18. Estructura de las clases de un agente. 30

Figura 19. Código swt 1. 31

Figura 20. Código swt 2. 31

Figura 21. Diagrama funcionamiento SWT. 31

Figura 22. Diagrama funcionamiento GridLayout. 32

Figura 23. Diseño autómata 33

Figura 24. Paquetes del agente. 33

Figura 25. Archivo automata.xml 34

Figura 26. Acciones Semánticas del Agente 34

Figura 27. Paquetes para Recursos 35

Figura 28. Ejemplo real de las clases de dominio del agente Médico..... 36

Figura 29. Envío evento recurso –agente 37

Figura 30. Envío evento agente –agente 37

Figura 31. Parte de la descripción de la organización del sistema. 38

Figura 32. Comunicación centralizada o distribuida con la base de datos. 40

Figura 33. Funcionamiento RMI 44

Figura 34. Gráfico métricas. 46

Anexo I: Casos de uso.

CU–01	**Autentificar al usuario**
Objetivo en contexto	Acceder al sistema.
Precondiciones	Debe ser posible establecer la conexión con la base de datos.
Postcond. si éxito	Se accede al sistema.
Postcond. si fallo	Se piden de nuevo los datos.
Actores	Usuario.
Secuencia normal	1 – Introducir número de usuario. 2 – Introducir contraseña. 3 – Aceptar. 4 – Acceder al sistema.
Secuencias alternativas	3a – Si el usuario no está en el sistema da un error y se da la opción a introducir de nuevo los datos. 3b – Si la contraseña no corresponde al usuario da un error y se da la opción a introducir de nuevo los datos.

CU–02	**Establecer estado de un paciente en consulta**
Objetivo en contexto	Asignar estado a un paciente que se encuentra en consulta.
Precondiciones	El paciente debe estar presente en la agenda actual. Conexión con base de datos.
Postcond. si éxito	Se establece el estado de un determinado paciente.
Postcond. si fallo	No se producen cambios. Volvemos a pantalla inicial
Actores	Usuario.
Secuencia normal	1 – Seleccionar un paciente. 2 – Asignar nuevo estado. 3 – Se asigna un color preestablecido según el estado a la posición donde se ubica el paciente.
Secuencias alternativas	

CU–03	**Calcular tiempo estimado de entrada**
Objetivo en contexto	A partir de un paciente establecer cuanto tiempo tardaría en pasar a consulta según el estado actual de retraso.
Precondiciones	El paciente debe estar presente en la agenda actual. Conexión con base de datos
Postcond. si éxito	Devuelve el tiempo aproximado que tardará en pasar a consulta.
Postcond. si fallo	Se vuelve a pantalla inicial.
Actores	Usuario.
Secuencia normal	1 – Seleccionar un paciente de la agenda actual 2 – Calcular tiempo de retraso a partir de la situación actual de consulta 3 – Devuelve dicho valor
Secuencias alternativas	2a – El sistema muestra una ventana de error y se vuelve a la ventana inicial de la agenda

CU–04	Añadir nuevo Usuario (profesional)
Objetivo en contexto	Añadir nuevo profesional al sistema, con su propia configuración de agenda.
Precondiciones	Debe ser posible establecer la conexión con la base de datos.
Postcond. si éxito	Se crea un nuevo profesional.
Postcond. si fallo	No se producen modificaciones.
Actores	Usuario.
Secuencia normal	1 – Introducir datos personales relativos al nuevo profesional. 2 – Introducir datos de configuración de la agenda diaria y periodo vacaciones. 3 – Establecer nombre de usuario y contraseña que asignado al nuevo profesional que se crea 4 - Aceptar 5– Creación de nuevo usuario del sistema en la base de datos.
Secuencias alternativas	3a – Si el usuario introducido ya esta siendo usado por otra persona, da un mensaje de error y se da la opción de introducir de nuevo los datos. 5a - Si se produce algún error de conexión con la base de datos aparece un mensaje de error y pide que reinicie la operación desde el principio. En este caso los cambios no se han guardado.

CU–05	Modificar usuario
Objetivo en contexto	Modificar los datos personales de un usuario o su configuración de la agenda.
Precondiciones	Debe ser posible establecer la conexión con la base de datos.
Postcond. si éxito	Se modifica parte de la información almacenada de un usuario.
Postcond. si fallo	No se producen cambios en el usuario.
Actores	Usuario.
Secuencia normal	1 – Modificar datos del formulario del usuario o de la configuración de la agenda 2 – Guardar 3 – Datos del usuario modificados en la base de datos.
Secuencias alternativas	2a – Si se modifica el nombre de usuario y este ya está siendo usado por otra persona, da un mensaje de error y se da la opción de introducir de nuevo los datos. 3b - Si se produce algún error de conexión con la base de datos aparece un mensaje de error y pide que reinicie la operación desde el principio. En este caso los cambios no se han guardado.

CU–06	Eliminar usuario
Objetivo en contexto	Borrar un usuario del sistema.
Precondiciones	Debe ser posible establecer la conexión con la base de datos.
Postcond. si éxito	Se borra un usuario del sistema. Pero no su historial.
Postcond. si fallo	Mensaje de error. No se producen cambios.
Actores	Usuario.
Secuencia normal	1 – Opción de eliminar usuario. 2 – Confirmar orden de eliminación. 3 – Aceptar. 4 – Borrado del usuario de la base de datos. Borrado de la agenda preestablecida para dicho usuario. 5 - Si tenía citas pendientes para el futuro aparecerá un mensaje con los pacientes que deben ser reubicados o avisados.
Secuencias alternativas	3a - Si el usuario compartía actividades con otros usuarios, aparecerá un mensaje de error que indique que deben eliminarse estas, antes de eliminar definitivamente el usuario. La operación queda anulada. 4a - Si se produce un error en la base de datos, aparecerá un mensaje de error y se pedirá que se repita la operación.

CU–07	Pedir hora
Objetivo en contexto	Un paciente pide hora para consulta.
Precondiciones	Debe ser posible establecer la conexión con la base de datos.
Postcond. si éxito	Se añade el paciente a la agenda.
Postcond. si fallo	Se pide nueva fecha para la cita.
Actores	Paciente.
Secuencia normal	1 – Se selecciona una fecha y hora. 2 – Se introducen los datos del paciente. 3 – Aceptar.
Secuencias alternativas	3a – Si el paciente está en el sistema se busca y se selecciona. 3b – Si el paciente no está en el sistema se le crea ficha y se selecciona. **CU-08**

CU–08	Crear una ficha
Objetivo en contexto	Crear una ficha para un paciente nuevo.
Precondiciones	Debe ser posible establecer la conexión con la base de datos.
Postcond. si éxito	El paciente se añade al sistema.
Postcond. si fallo	Mensaje de error y se piden de nuevo los datos erróneos.
Actores	Paciente.
Secuencia normal	1 – Introducir los datos que pide el sistema. 2 – Aceptar.
Secuencias alternativas	2a – Si el paciente ya está en el sistema se vera una pantalla de error. 2b – Si los datos introducidos son incorrectos se mostrará un error.

CU–09	Consultar una ficha
Objetivo en contexto	Consultar la ficha de un paciente.
Precondiciones	Debe ser posible establecer la conexión con la base de datos.
Postcond. si éxito	Se obtienen los datos del paciente.
Postcond. si fallo	Mensaje de error.
Actores	Paciente.
Secuencia normal	1 – Introducir el nombre y/o apellidos del paciente en cuestión. 2 – Aceptar.
Secuencias alternativas	2a – Si el paciente no está en el sistema se verá una pantalla de error.

CU–10	Modificar una ficha
Objetivo en contexto	Se cambian los datos de un paciente existente.
Precondiciones	Debe ser posible establecer la conexión con la base de datos.
Postcond. si éxito	Se cambian los datos del paciente en cuestión.
Postcond. si fallo	Mensaje de error y se piden de nuevo los datos erróneos.
Actores	Paciente.
Secuencia normal	1 – Introducir los datos que pide el sistema. 2 – Aceptar.
Secuencias alternativas	2a – Si los datos introducidos son incorrectos se mostrara un error.

CU–11	Buscar visitas por paciente
Objetivo en contexto	Buscar el historial de visitas del paciente.
Precondiciones	Debe ser posible establecer la conexión con la base de datos.
Postcond. si éxito	Se obtiene un listado con las visitas del paciente en cuestión.
Postcond. si fallo	Mensaje de error.
Actores	Paciente.
Secuencia normal	1 – Introducir el nombre y/o apellidos del paciente en cuestión. 2 – Aceptar.
Secuencias alternativas	2a – Si el paciente no está en el sistema da un error.

CU–12	Atender a un paciente
Objetivo en contexto	Atender a un paciente.
Precondiciones	El paciente tiene una cita fijada.
Postcond. si éxito	El paciente queda atendido y vuelve a casa contento.
Postcond. si fallo	Se piden de nuevo los datos.
Actores	Paciente y Doctor.
Secuencia normal	1 - Buscar al paciente en cuestión. 2 - Introducir datos primera fase: Motivo de consulta, Descripción enfermedad actual y Antecedentes. 3 - Hacer clic en siguiente. 4 - Introducir datos de segunda fase: Analítica, añadir documentos e imágenes, diagnóstico. 5 - Hacer clic en siguiente. 6 - Introducir datos tercera fase: Tratamientos. 7 - Finalizar.
Secuencias alternativas	1a - Si el paciente no está en el sistema crear una ficha nueva. **CU-08** 2a, 4a, 6a - Si el sistema da error volver a introducir los datos.

CU–13	Crear un consentimiento
Objetivo en contexto	Crear un consentimiento.
Precondiciones	Debe ser posible establecer la conexión con la base de datos.
Postcond. si éxito	El consentimiento se añade al sistema.
Postcond. si fallo	Mensaje de error y se piden de nuevo los datos erróneos.
Actores	
Secuencia normal	1 – Introducir los datos que pide el sistema. 2 – Aceptar.
Secuencias alternativas	2a – Si el consentimiento ya está en el sistema se verá una pantalla de error. 2b – Si los datos introducidos son incorrectos se mostrará un error.

CU–14	Modificar un consentimiento
Objetivo en contexto	Se cambian los datos de un consentimiento existente.
Precondiciones	Debe ser posible establecer la conexión con la base de datos.
Postcond. si éxito	Se cambian los datos del consentimiento en cuestión.
Postcond. si fallo	Mensaje de error y se piden de nuevo los datos erróneos.
Actores	
Secuencia normal	1 – Introducir los datos que pide el sistema. 2 – Aceptar.
Secuencias alternativas	2a – Si los datos introducidos son incorrectos se mostrará un error.

CU–15	Imprimir un consentimiento
Objetivo en contexto	Se imprime un consentimiento.
Precondiciones	Debe ser posible establecer la conexión con la base de datos.
Postcond. si éxito	La impresora recibe la orden de imprimir el consentimiento.
Postcond. si fallo	Mensaje de error.
Actores	
Secuencia normal	1 – Buscar el consentimiento en cuestión. 2 – Aceptar.
Secuencias alternativas	2a – Si el consentimiento no está en el sistema se muestra un mensaje de error.

CU–16	Modificar un Historial
Objetivo en contexto	Se cambian los datos de un paciente existente.
Precondiciones	Debe ser posible establecer la conexión con la base de datos. Debe existir el paciente.
Postcond. si éxito	Se cambian los datos del paciente en cuestión.
Postcond. si fallo	Mensaje de error y se piden de nuevo los datos erróneos.
Actores	Paciente.
Secuencia normal	1 – Modificar los datos que pide el sistema. 2 – Aceptar.
Secuencias alternativas	2a – Si los datos introducidos son incorrectos se mostrará un error.

CU–17	Consultar un Historial
Objetivo en contexto	Consultar el historial de un paciente.
Precondiciones	Debe ser posible establecer la conexión con la base de datos. Debe existir el paciente.
Postcond. si éxito	Se obtienen los datos del paciente.
Postcond. si fallo	Mensaje de error.
Actores	Paciente.
Secuencia normal	1 – Introducir el nombre y/o apellidos del paciente en cuestión. 2 – Aceptar. 3 - Se muestran los datos correspondientes al paciente buscado
Secuencias alternativas	3a – Si el paciente no está en el sistema se verá una pantalla de error.

CU–18	Consultar un medicamento
Objetivo en contexto	Consultar información adicional relativa a un medicamento.
Precondiciones	Debe ser posible establecer la conexión con la base de datos.
Postcond. si éxito	Se obtienen los datos correspondientes al medicamento que buscamos.
Postcond. si fallo	Mensaje de error.
Actores	Paciente.
Secuencia normal	1 – Introducir datos de búsqueda del medicamento (nombre, principio activo, enfermedad asociada, etc.). 2 – Aceptar. 3 - Se muestran los datos correspondientes a los medicamentos que se corresponden con los criterios de búsqueda seleccionados.
Secuencias alternativas	3a – Si no existe ningún medicamento que cumpla dichas características da un mensaje de aviso.

CU–19	Consultar una enfermedad
Objetivo en contexto	Consultar información adicional relativa a una enfermedad.
Precondiciones	Debe ser posible establecer la conexión con la base de datos.
Postcond. si éxito	Se obtienen los datos correspondientes a la enfermedad que buscamos.
Postcond. si fallo	Mensaje de error.
Actores	Paciente.
Secuencia normal	1 – Introducir datos de búsqueda de la enfermedad (nombre, síntomas, etc.). 2 – Aceptar. 3 - Se muestran los datos correspondientes a las enfermedades que se corresponden con los criterios de búsqueda seleccionados.
Secuencias alternativas	3a – Si no existe ninguna enfermedad que cumpla dichas características da un mensaje de aviso.

CU–20	Crear una receta
Objetivo en contexto	Crear una receta.
Precondiciones	Debe ser posible establecer la conexión con la base de datos.
Postcond. si éxito	La receta se añade al sistema.
Postcond. si fallo	Mensaje de error y se piden de nuevo los datos erróneos.
Actores	
Secuencia normal	1 – Introducir los datos que pide el sistema. 2 – Aceptar.
Secuencias alternativas	2a – Si la receta ya está en el sistema se verá una pantalla de error. 2b – Si los datos introducidos son incorrectos se mostrará un error.

CU–21	Modificar de una receta
Objetivo en contexto	Cambiar los datos de una receta existente.
Precondiciones	Debe ser posible establecer la conexión con la base de datos.
Postcond. si éxito	Se cambian los datos de la receta en cuestión.
Postcond. si fallo	Mensaje de error y se piden de nuevo los datos erróneos.
Actores	
Secuencia normal	1 – Introducir los datos que pide el sistema. 2 – Aceptar.
Secuencias alternativas	2a – Si los datos introducidos son incorrectos se mostrará un error.

CU–22	Imprimir una receta
Objetivo en contexto	Se imprime una receta.
Precondiciones	Debe ser posible establecer la conexión con la base de datos.
Postcond. si éxito	La impresora recibe la orden de imprimir la receta.
Postcond. si fallo	Mensaje de error.
Actores	
Secuencia normal	1 – Buscar la receta en cuestión. 2 – Aceptar.
Secuencias alternativas	2a – Si la receta no esta en el sistema se muestra un mensaje de error.

CU–23	Imprimir una receta desde la Web
Objetivo en contexto	Se imprime una receta online.
Precondiciones	Disponer de conexión a Internet.
Postcond. si éxito	La impresora recibe la orden de imprimir la receta.
Postcond. si fallo	Mensaje de error.
Actores	
Secuencia normal	1 – Entrar a la web con el nombre de usuario y contraseña. 2 – Buscar la receta. 3 - Imprimir.
Secuencias alternativas	1a - Si no existe el nombre de usuario o la contraseña es errónea se muestra un error. 2a – Si la receta no está en el sistema se muestra un mensaje de error.

CU–24	**Consultar el estado de la consulta desde la Web**
Objetivo en contexto	Obtener información sobre posibles retrasos en la consulta.
Precondiciones	Disponer de conexión a Internet.
Postcond. si éxito	Se muestra la información.
Postcond. si fallo	Mensaje de error.
Actores	Paciente.
Secuencia normal	1 – Entrar a la web con el nombre de usuario y contraseña. 2 – Hacer clic en Estado de Consulta. 3 - Aceptar.
Secuencias alternativas	1a - Si no existe el nombre de usuario o la contraseña es errónea se muestra un error.

CU–25	**Ver próximas citas desde la Web**
Objetivo en contexto	Obtener información sobre las citas concertadas.
Precondiciones	Disponer de conexión a Internet.
Postcond. si éxito	Se muestra la información.
Postcond. si fallo	Mensaje de error.
Actores	Paciente.
Secuencia normal	1 – Entrar a la web con el nombre de usuario y contraseña. 2 – Hacer clic en Próximas Citas. 3 - Aceptar.
Secuencias alternativas	1a - Si no existe el nombre de usuario o la contraseña es errónea se muestra un error. 2a – Si la receta no está en el sistema se muestra un mensaje de error.

CU–26	Enviar mensaje al médico / administración desde la Web
Objetivo en contexto	Enviar un mensaje al médico / administración desde Internet.
Precondiciones	Disponer de conexión a Internet.
Postcond. si éxito	Envío de mensaje al médico / administración.
Postcond. si fallo	Mensaje de error.
Actores	Paciente.
Secuencia normal	1 – Entrar a la web con el nombre de usuario y contraseña. 2 – Hacer clic en mensajes. 3 - Añadir texto del mensaje 4 - Aceptar.
Secuencias alternativas	1a - Si no existe el nombre de usuario o la contraseña es errónea se muestra un error.

CU–27	Subir documentos desde la Web
Objetivo en contexto	Se añaden documentos al del paciente subiéndolos a través de la Web
Precondiciones	Disponer de conexión a Internet.
Postcond. si éxito	Se guardan en el historial del paciente los documentos subidos.
Postcond. si fallo	Mensaje de error.
Actores	Paciente.
Secuencia normal	1 – Entrar a la web con el nombre de usuario y contraseña. 2 – Hacer clic en subir documentos. 3 - Añadir los documentos que se quieren subir 4 - Añadir texto explicativo si se precisa 5 - Aceptar.
Secuencias alternativas	1a - Si no existe el nombre de usuario o la contraseña es errónea se muestra un error.

CU–28	Consultar una ficha desde la Web
Objetivo en contexto	Consultar la ficha de un paciente desde la web.
Precondiciones	Disponer de conexión a Internet.
Postcond. si éxito	Se obtienen los datos del paciente.
Postcond. si fallo	Mensaje de error.
Actores	Paciente.
Secuencia normal	1 - Entrar a la web con el nombre de usuario y contraseña. 2 - Hacer clic en ficha.
Secuencias alternativas	1a - Si no existe el nombre de usuario o la contraseña es errónea se muestra un error.

CU–29	Modificar una ficha desde la Web
Objetivo en contexto	Modificar la ficha de un paciente desde la web.
Precondiciones	Disponer de conexión a Internet.
Postcond. si éxito	Se modifican los datos del paciente.
Postcond. si fallo	Mensaje de error.
Actores	Paciente.
Secuencia normal	1 - Entrar a la web con el nombre de usuario y contraseña. 2 - Hacer clic en ficha. 3 - Modificar los datos que se quieran. 4 – Guardar.
Secuencias alternativas	1a - Si no existe el nombre de usuario o la contraseña es errónea se muestra un error.

Anexo II: Métricas Investigación

Fecha	Investigación ICARO	Horas	Notas
Noviembre 09	Investigación de como funciona la aplicación de acceso	4	Un poco lioso ver como se construía el agente y como era la comunicación con eventos
Noviembre 09	Creación de un agente de ejemplo	8	Costó mucho que funcionara, el agente se atascaba, no se sabía porque
Diciembre 09	Creación de un recurso de visualización con SWT	10	ICARO y SWT dan muchos problemas juntos. Hay que tratar los threads SWT con mucho cuidado. Fue complicado dar con la solución. Esto nos siguió dando muchos problemas mas adelante
Diciembre - Enero	Problemas de compatibilidad entre SWT e ICARO	8	Hay funcionalidad de SWT que no hemos podido llegar a usar debido a que da problemas con ICARO
Febrero 09	Creación de los templates para la creación de agentes nuevos	6	Una vez entendido como funcionaba ICARO optamos por hacer plantillas de los agentes con un recurso de visualización. Esto nos facilitó muchísimo la creación de nuevos agentes, nos permitía ahorrar tiempo y reducía el numero de fallos posibles
Febrero 09	Comunicación Agente - Recurso	6	Cogerle el truco a los eventos costó lo suyo. No estaba muy claro como obtener la interfaz de uso y además era muy sensible a los nombres
Febrero 09	Comunicación Agente - Agente sin enviar datos entre ellos	6	Problemas con la configuración del XML con varios agentes. Además, un pequeño error en un agente hacía que se bloqueara todo
Marzo 09	Comunicación Agente - Agente con datos	3	No entendimos muy bien como se "transformaban" los datos al enviarlos en los eventos. ICARO transforma los parámetros en un único Object[] (Object si solo había un parámetro)
Marzo 09	Acceso a la persistencia	4	Se trata de eventos síncronos a diferencia de la comunicación Agente-Agente que es asíncrona
Abril 09	Probar IcaroMini 1.2.1	3	No llego a funcionar por un problema con el paquete jaxb
Mayo 09	Migración a IcaroMini 1.3.1	3	Esta vez si funcionó bien la migración. Tuvimos que renombrar varias cosas básicas de ICARO
Mayo 09	Acceder a ICARO remotamente usando RPC (también requería investigación de GWT)	8	Se ha conseguido crear un JAR de ICARO pero no logramos arrancar ICARO remotamente. Hay un problema con log4j. No se ha conseguido que funcione.
Mayo 09	Investigación del funcionamiento de los servlets y su integración con ICARO	10	Se ha estudiado el funcionamiento de los servlets pero su integración con ICARO no ha funcionado.
Mayo 09	Investigación del funcionamiento de RMI y su integración con ICARO	10	Se ha investigado el funcionamiento de RMI pero no se vio claro como se integraría con ICARO. Suponía adaptar todos los agentes.

Anexo III: Pruebas

Test-Id	Medico-01
Descripción	El médico elige un día para consultar los pacientes que tiene.
Entrada	Ninguna
Actores	Médico
Objetivo	Mostrar una lista en pantalla con los pacientes del día.
Fallos posibles	1. Fecha incorrecta. 2. Problema de conexión a la BD. 3. Muestra un resultado incorrecto.
Pasos para la prueba. Acciones	
Id	**Descripción**
1	Hacer Login como médico.
2	Seleccionar una fecha en el calendario haciendo doble clic.
3	Mirar la lista de pacientes y compararla con la lista que se esperaba.
Ejecución de la prueba	
Resultado	
Comentarios	
Fecha	
Autor de la prueba	

Test-Id	Medico-02
Descripción	El médico consulta el historial de un paciente.
Entrada	Ninguna
Actores	Médico
Objetivo	Ver la ventana de historial con los datos del paciente seleccionado.
Fallos posibles	1. No se encuentra el historial en la BD. 2. Se muestra el historial de otro paciente. 3. Da algún error al abrir la ventana de historial.
Pasos para la prueba. Acciones	
Id	**Descripción**
1	Hacer Login como médico.
2	Seleccionar un paciente de la lista de citas o de la lista de búsqueda.
3	Pulsar en el botón Historial.
4	Comprobar que los datos mostrados en la nueva ventana sean correctos.
Ejecución de la prueba	
Resultado	
Comentarios	
Fecha	
Autor de la prueba	

Test-Id	Medico-03
Descripción	El médico modifica el historial de un paciente.
Entrada	Ninguna
Actores	Médico
Objetivo	Modificar el historial de un paciente.
Fallos posibles	1. Error de conexión con la BD. 2. Error al guardar los datos en la BD. 3. Error al modificar alguno de los elementos del historial. 4. La ventana del historial no deja guardar los cambios.
Pasos para la prueba. Acciones	
Id	**Descripción**
1	Hacer Login como médico.
2	Seleccionar un paciente, abrir su historial y seleccionar una visita concreta.
3	Cambiar el texto de alguno de los cambios. Añadir o borrar algún documento, añadir o borrar algún medicamento.
4	Hacer clic en guardar.
5	Cerrar la visita, volver a abrirla y comprobar que los datos se han guardado.
Ejecución de la prueba	
Resultado	
Comentarios	
Fecha	
Autor de la prueba	

Test-Id	Medico-04
Descripción	El médico añade un medicamento nuevo.
Entrada	Nombre, Principio Activo, Descripción e Indicaciones.
Actores	Médico
Objetivo	Añadir un nuevo medicamento al sistema desde la ventana de médico.
Fallos posibles	1. El medicamento ya se encuentra en la base de datos pero el sistema no se da cuenta. 2. No se inserta bien el medicamento. 3. La ventana falla a la hora de intentar guardar el medicamento. 4. El medicamento se inserta pero no sale en la lista.
Pasos para la prueba. Acciones	
Id	**Descripción**
1	Hacer Login como médico.
2	Seleccionar la pestaña medicamentos.
3	Pulsar en el botón Añadir medicamento.
4	Rellenar los campos. Se puede dejar alguno vacío para ver si avisa de que faltan por rellenar campos.
5	Dar al botón Guardar y comprobar si se actualiza la lista de medicamentos.
Ejecución de la prueba	
Resultado	
Comentarios	
Fecha	

Autor de la prueba	

Test-Id	Login
Descripción	Autentificación de usuario. Mostrar al final la ventana del usuario.
Entrada	Nombre de usuario y contraseña.
Actores	-
Objetivo	Acceder a la BBDD encontrar el usuario y mostrar su ventana inicial.
Fallos posibles	4. El usuario no existe 5. No se introduce alguno de los dos campos 6. Muestra una ventana de usuario que no corresponde 7. Si cancelar no sale de la aplicación
Pasos para la prueba. Acciones	
Id	**Descripción**
1	Arranque del sistema
2	Introducir usuario y contraseña erróneos
3	Dejar uno de los campos o los dos sin rellenar
4	Introducir distintos usuarios y contraseñas validas para distintos tipos de usuario.
5	Cancelar el login
Ejecución de la prueba	
Resultado	
Comentarios	
Fecha	
Autor de la prueba	

Test-Id	ConsultarProximasCitas_Sec
Descripción	Desde la ventana de un usuario secretaria consultar las próximas citas de una determinado paciente
Entrada	Nombre y/o teléfono de un paciente
Actores	Secretaria
Objetivo	Acceder a la BBDD. Comprobar que existe dicho paciente y devolver si hubiere las citas que tiene dicho paciente previstas con fecha posterior al día actual.
Fallos posibles	4. Introducir un paciente repetido. Cual muestra? 5. Habiendo introducido un paciente correctamente con citas posteriores a hoy, no muestre nada 6. Con un paciente correcto muestre fechas anteriores a la de hoy 7. mostrar incorrectamente los datos 8. Dejar en blanco alguno de los campos. (puede haber pacientes iguales y no sabe cual elegir) 9. Botones que no actúan adecuadamente
Pasos para la prueba. Acciones	
Id	**Descripción**
1	Introducir solo un campo + buscar
2	Introducir ambos campos incorrectamente + buscar
3	Introducir ambos campos correctamente +buscar
4	Cancelar acción

Ejecución de la prueba	
Resultado	
Comentarios	
Fecha	
Autor de la prueba	

Test-Id	Copiar/Pegar/Borrar_Cita_Sec
Descripción	A partir de una cita ya rellena o no debe poder copiar y/o pegar en cualquier otro sitio o borrarla. Cambiando los datos en la persistencia y asignando una nueva cita si procede
Entrada	Ventana Secretaria con la Agenda visible
Actores	Secretaria
Objetivo	Poder copiar solo las entradas de la agenda ocupadas. Borrar una entrada de una agenda accediendo a la bbdd. Pegar algo previamente copiado sin errores y que se quede reflejado en la bbdd
Fallos posibles	1. Copiar no tenga el efecto visible o en BBDD 2. Pegar no tenga efecto visible o en BBDD 3. Intente pegar algo q no ha copiado 4. Borrar no tenga efecto visible o en BBDD 5. En cualquiera de los casos anteriores que no guarde correctamente los datos en la BBDD
Pasos para la prueba. Acciones	
Id	**Descripción**
1	Cargar una agenda con citas rellenas
2	Elegir una vacía. Seleccionar pegar + copiar +pegar + borrar
3	Copiar sobre vacío (Se queda almacenado lo copiado anteriormente)
4	Elegir una entrada de la agenda rellena. Seleccionar pegar + copiar + pegar + borrar
Ejecución de la prueba	
Resultado	
Comentarios	
Fecha	
Autor de la prueba	

Test-Id	CambiarFechaAgenda_Sec
Descripción	Desde la ventana de agenda y un día cualquiera se cambia a un menú con calendarios que permiten cambiar la fecha de la agenda que se muestra.
Entrada	Ventana Secretaria con la Agenda visible
Actores	Secretaria
Objetivo	Comprobar que se produce un cambio de fecha en la agenda. Se accede correctamente a la BBDD y se extraen los datos correctos de la agenda en el día que se ha seleccionado. Comprobar que todos los botones relacionados con esta acción tienen el efecto esperado.
Fallos posibles	1. Alguno de los botones falla. Si esto ocurre es probable que aunque se continúe con la operación los datos mostrados sean erróneos 2. Muestra una fecha incorrecta. 3. Aparece una agenda vacía. No accede correctamente a la BBDD 4. Fallo por excepción
Pasos para la prueba. Acciones	

Id	Descripción
1	Clic "cambio Fecha" + Seleccionar una fecha (doble clic) del calendario de arriba.
2	Comprobar para cada pestaña de médico que los datos que se han cargado en la agenda son correctos.
3	Comprobar el horario de mañana y de tarde con los datos correctos.
4	Clic "cambio Fecha" + Seleccionar una fecha (doble clic) del calendario de abajo.
5	Comprobar para cada pestaña de médico que los datos que se han cargado en la agenda son correctos.
6	Comprobar el horario de mañana y de tarde con los datos correctos.
7	Clic "Agenda Hoy". Comprobar agenda del día de hoy.
Ejecución de la prueba	
Resultado	
Comentarios	
Fecha	
Autor de la prueba	

Test-Id	MostrarFichaPaciente_Sec
Descripción	Desde la agenda de la secretaria se puede seleccionar un paciente y posteriormente hacer clic sobre "ficha". Puede ser que la entrada seleccionada este vacía o no se haya seleccionado ninguna. También puede ser que el paciente no tenga ninguna ficha todavía. En cuyo caso debe comprobar si se desea crear.
Entrada	Ventana Secretaria con la Agenda visible
Actores	Secretaria
Objetivo	Comprobar que accede correctamente a la BBDD para consultar la ficha de un paciente. Comprobar que la acción trata correctamente los casos en los que no recibe los datos correctos.
Fallos posibles	1. Excepción si no se ha seleccionado ninguna entrada de la agenda previamente. 2. Excepción si la entrada seleccionada esta vacía. 3. Fallo si el paciente no tiene ficha o no existe en la bbdd. 4. Fallo con paciente de bbdd correcto y con ficha. No encuentra su ficha.
Pasos para la prueba. Acciones	
Id	**Descripción**
1	Clic sobre ficha sin seleccionar ninguna entrada
2	Seleccionar una entrada vacía de la agenda y seleccionar ficha
3	Seleccionar una entrada de un paciente sin ficha y seleccionar ficha
4	Cancelar la creación de la ficha
5	Seleccionar una entrada de un paciente sin ficha y seleccionar ficha
6	Aceptar la creación + guardar
7	Seleccionar una entrada de un paciente con ficha y seleccionar ficha
Ejecución de la prueba	
Resultado	
Comentarios	
Fecha	
Autor de la prueba	

Autorización a la UCM

Por la presente se autoriza a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto esta memoria, como el código, la documentación y el prototipo desarrollado.

Autores:

Camilo Andrés Benito Rojas
Dulce María Valerón Almazán